2013

EFFICIENT MULTIPLIER-LESS VLSI ARCHITECTURES FOR FOLDED PIPELINED COMPLEX FFT CORE

ANSUMAN DIPTISANKAR DAS
DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGYROURKELA

EFFICIENT MULTIPLIER-LESS VLSI ARCHITECTURES FOR FOLDED PIPELINED COMPLEX FFT CORE

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology

In

VLSI DESIGN AND EMBEDDED SYSTEM

By

Ansuman DiptiSankar Das

Department of Electronics and Communication Engineering

National Institute Of Technology

Rourkela

2011 – 2013

EFFICIENT MULTIPLIER-LESS VLSI ARCHITECTURES FOR FOLDED PIPELINED COMPLEX FFT CORE

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology
In
VLSI DESIGN AND EMBEDDED SYSTEM

by

Ansuman DiptiSankar Das

211EC2077

Under the Guidance of
Prof. K. K. Mahapatra

Department of Electronics and Communication Engineering

National Institute Of Technology

Rourkela

2011 – 2013

To my mother, bhala dei, aradhana, and my late grandfather.

Some of them have left me, some of them have abandoned me, but still it's their constant support, motivation and inspiration that has made life beautiful.

DEPARTMENT OF ELECTRONICS AND
COMMUNICATION ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY, ROURKELA
ORISSA, INDIA-769008

CERTIFICATE

This is to certify that the Thesis Report entitled "**EFFICIENT MULTIPLIER-LESS VLSI ARCHITECTURES FOR FOLDED PIPELINED COMPLEX FFT CORE**", submitted by **Mr. ANSUMAN DIPTISANKAR DAS bearing roll no. 211EC2077** in partial fulfilment of the requirements for the award of **Master of Technology in Electronics and Communication Engineering** with specialization in "**VLSI Design and Embedded Systems**" during session 2011 - 2013 at National Institute of Technology, Rourkela is an authentic work carried out by him under my supervision and guidance.

To the best of my knowledge, the matter embodied in the thesis has not been submitted to any other university/institute for the award of any Degree or Diploma.

Place: Rourkela

Date: 29TH May, 2013

Prof. (Dr.) K. K. MAHAPATRA
Dept. of E.C.E
National Institute of Technology
Rourkela – 769008

Acknowledgements

I would like to express my gratitude to my thesis guide **Prof. K. K. Mahapatra** for his guidance, advice and support throughout my thesis work. I am especially indebted to him for teaching me both research and writing skills, which have been proven beneficial for my current research and future career. Without his endless efforts, knowledge, patience, and answers to my numerous questions, this research would have never been possible. The experimental methods and results presented in this thesis have been influenced by him in one way or the other. It has been a great honour and pleasure for me to do research under supervision of Prof. K. K. Mahapatra. Working with him has been a great experience. I would like to thank him for being my advisor here at National Institute of Technology, Rourkela.

Next, I want to express my respects to **Prof. Samit Ari, Prof .A.K. Swain, Prof. D.P. Acharya, Prof. P. K. Tiwari, Prof. N. Islam, Prof. Sukadev Meher, Prof. S. K. Patra, Prof. S. K. Behera , Prof. Poonam Singh** for teaching me and also helping me how to learn. They have been great sources of inspiration to me and I thank them from the bottom of my heart.

I would like to thank to all my faculty members and staff of the Department of Electronics and Communication Engineering, N.I.T. Rourkela, for their generous help for the completion of this thesis.

I would like to thank all my friends and especially my classmates for thoughtful and mind stimulating discussions we had, which prompted to think beyond the obvious. I've enjoyed their companionship so much during my stay at NIT, Rourkela.

I am especially indebted to my mother for her love, sacrifice, and support. She is my first teachers, after I came to this world and I have set of great examples for me about how to live, study and work. I am grateful to her for guiding my steps on the path of achievements since my infanthood.

Ansuman DiptiSankar Das

Abstract

Fast Fourier transform (FFT) has become ubiquitous in many engineering applications. FFT is one of the most employed blocks in many communication and signal processing systems. Efficient algorithms are being designed to improve the architecture of FFT. Higher radix FFT algorithms have the traditional advantage of using less number of computational elements and are more suitable for calculating FFT of long data sequence. Among the different proposed algorithms, the split-radix FFT has shown considerable improvement in terms of reducing hardware complexity of the architecture compared to radix-2 and radix-4 FFT algorithms. Here radix-4, radix-8, and split-radix algorithms have been used in the design of different proposed complex FFT cores.

The growing popularity of adopting virtual instrumentation (modular, customizable, software-defined instrumentation) has only became possible due to the use of LabVIEW with a highly interactive process known as graphical system design. The CompactRIO programmable automation controller is an advanced embedded control and data acquisition system designed for applications that require high performance and reliability. The work explains the real-time implementation of 256-point FFT and finding the power spectrum using LabVIEW and CompactRIO. The proposed implementation uses only 3077 slices(21%), 2489 slice registers(8.7%), 4651 slice LUTs(16.2%) on a 400 MHz real time embedded processor.

New distributed arithmetic (NEDA) is one of the most used techniques in implementing multiplier-less architectures of many digital systems. One of the designs proposes efficient multiplier-less VLSI architectures of split-radix FFT algorithm using NEDA. As the proposed architecture does not contain any multiplier block, substantial reduction in terms of power and area can greatly be observed at a higher speed. One of the proposed architectures in this section is designed by considering all the inputs at a time and the other is designed by considering 4 inputs at a time. The total number of inputs is 32 in both designs. The proposed designs have implemented in both FPGA as well as ASIC design flows. 180nm process technology is being used for ASIC implementation. The results show the improvements of proposed designs compared to the other existing architectures.

The next two architectures compute 64-points complex Fast Fourier Transform (FFT) using radix-4 and radix-8 algorithm respectively. The proposed architectures have been implemented using Xilinx ISE. The proposed architectures have also been implemented on

ASIC in 0.18µm process technology using Synopsys DC for logic synthesis and Cadence SoC Encounter for physical design. The process technology that has been followed to carryout physical design of the proposed architectures is UMC 0.18µm mixed mode generic core. The physical design of proposed architectures has been made in such a way that the timing constraints are met after both placement as well as routing. A comparison has been done for the proposed architectures in terms of area, speed, power, and device utilization. The results show the improvements of proposed designs compared to other existing designs. The proposed architecture for 64-points complex Fast Fourier Transform (FFT) using radix-4 can operate at a maximum frequency of 327.384 MHz while the proposed architecture for 64-points complex Fast Fourier Transform (FFT) using radix-8 can operate at a maximum frequency of 447.838 MHz on Xilinx Virtex-5 FPGAs. Finally, the architecture for a 512-point complex FFT core using radix-8 algorithm has been proposed.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Motivation

Problem Description

Organisation of this Thesis

Chapter 1

Introduction

1.1 Motivation

Fast Fourier Transforms (FFT) [1] – [2] is an algorithm for speedy calculation of Discrete Fourier transform (DFT) of an input data vector used in various signal and image processing applications [3]. The FFT is nothing but a DFT algorithm which reduces the number of computations needed for N points from $O(N^2)$ to $O(N \log N)$ where log is the base-2 logarithm using periodicity and property. Many FFT algorithms have been proposed to enhance the speed while reducing the area required and power consumption. Starting with the radix-2 Cooley-Tukey FFT algorithm [4], many FFT algorithms like radix-4, split-radix, radix-8, and radix-2^k FFT algorithms have been proposed [5] – [13]. Higher radix algorithms have the traditional advantage of using less number of computational elements and require a very simple structure for calculating FFT of long data sequence.

Many of the described FFT algorithms use multiplexers or ROMs or multipliers to compute twiddle multiplications [14] – [16]. The above algorithms use Distributed Arithmetic (DA) technique [17] and have the disadvantages of consuming more hardware and power. This leads the motivation towards the design of FFT algorithms which do not require any multipliers or ROMs or multiplexers. This can be possible by using New Distributed Arithmetic (NEDA) technique [18]. New Distributed Arithmetic (NEDA) is similar to DA except that it has no ROM unit, which is used in the case of DA. Thus, NEDA eliminates the use of multipliers and ROM to carry out multiplications in FFT. Implementation of FFT using NEDA improves performance of the system in terms of speed, area, and power. Some of the described FFT algorithms have used COordinate Rotation DIgital Computer (CORDIC) techniques to achieve less area and power consumption [19] - [21].

DSP system design techniques such as folding, pipelining have always improved performance of the systems in terms of hardware, latency, frequency, etc [22] – [25]. In DSP architectures, systematic control circuits are determined by using the folding transformation. In folding technique, time multiplexing of algorithm operations is done, by reducing to a single functional unit. Thus, in DSP systems, folding technique provides a means of trading time for area. Conventional folding technique can be used to reduce the number of hardware functional units by a factor of N at the expense of increasing the computation time or

multiplexing time by a factor of N. This technique also helps in data allocation in the required registers. To avoid excess amount of registers that are generated in these architectures while folding, there are techniques to minimize the number of registers needed to implement DSP architectures through folding.

Unlike the fixed, vendor-specific integrated circuits (ASIC) chips, an FPGA can be configured and reconfigured for different applications which provide precise timing and synchronization, simultaneous execution of parallel task, and rapid decision making. The growing popularity of virtual instruments which is the combination of user-defined software and modular hardware that implements custom systems with components for data acquisition, processing or analysis and presentation has became possible due to the graphical system design provided by LabVIEW. Historically, FPGA designers were forced to learn and use complex design languages such as Verilog or VHDL to program the FPGAs. Now, the problem can be solved by using LabVIEW and CompactRIO tools to program for self-customized FPGAs [26] – [29].

1.2 Problem Description

Fast Fourier Transform (FFT) is used to build various image processing systems and application specific Digital Signal Processing (DSP) hardware. Currently almost all proposed designs for FFT use ROMs or memory for complex twiddle multiplications. Proper techniques must be followed to eliminate the need of multipliers in FFT design. One of the most frequently used and significant method to eliminate the multipliers used in FFT design is using New Distributed Arithmetic (NEDA) for twiddle multiplications. While using NEDA technique, one must do precise shifting to reduce the number of adders.

While implementing FFT cores for long data sequence on FPGA, the number of bonded inputs and outputs (IOBs) are always a matter of concern. Precise techniques must be used to lessen the number of IOBs. We can reduce the number of IOBs if we can divide the long data sequence into a group of short data sequences of same length. By this not only throughput will be increased but also the number of IOBs will be reduced by a great factor. This can be achieved by folding technique. But by using folding technique, the computational time will be increased in a linear manner as well as the latency. To avoid excess amount of registers that are generated in these architectures while folding, there are techniques to minimise the number of registers needed to implement DSP architectures through folding.

Various proposed FFT algorithms have used pipelining in order to increase throughput and speed. This current work has used both folding and pipelining so that speed and throughput can be increased while not effecting the computational time. In a normal parallel design, the outputs starts coming after a very few clock cycles. The number of clock cycles to get the output increases as we incorporate pipelining and folding technique. Again while using pipelining and folding technique, there may be clock mismatch giving rise to undefined outputs in some clock cycles and wrong outputs in the other clock cycles. So, very strict attention must be paid while using folding and pipelining technique.

NEDA has been used in this work to design various FFT cores. Higher radix FFT algorithms also have been used to design the various FFT cores. Radix-4, split-radix and radix-8 butterfly sections has been optimized for better performance. The proposed designs in the present work are efficient in terms of area, speed and power. The present designs have been tested on various FPGAs. ASIC implementation of the proposed architectures in 0.18μm process technology using Synopsys DC for logic synthesis and Cadence SoC Encounter for physical design has also been conducted. The physical design of proposed architectures has been made in such a way that the timing constraints are met after both placement as well as routing.

The FFT algorithm has also been implemented on CompactRIO FPGA using LabVIEW programming. Prior to that, computation of FFT of an analog signal using LabVIEW and finding the power spectrum of an analog signal using LabVIEW has been conducted. The problems while acquiring data using CompactRIO from real world through different Input and Output (IO) modules has been solved successfully. Thorough testing of CompactRIO has been done in order to find the ideal operating conditions. The choice of sampling rates and successful synchronization between different modules of CompactRIO has also been done. Data transfer from FPGA to HOST has been done through Dynamic Memory Allocation (DMA) technique.

1.2 Organisation of this Thesis

This thesis has been divided into several chapters. A brief overview of the problem targeted in the current work has been presented in this chapter. The second chapter describes the various FFT algorithms used to design the FFT cores. The FPGA summary report and ASIC flow results are also given for the various butterfly structures used in the FFT core in the second chapter. The third chapter presents the overview of NEDA technique. The

CompactRIO architecture and a brief overview of the different components used have been discussed in chapter four. Chapter five presents four proposed architectures for different FFT cores:

- Real-time implementation of FFT using CompactRIO
- 32-Point Complex FFT Core Using Split-Radix Algorithm
- 64-Point Complex FFT Core Using Radix-4 Algorithm
- 64-Point Complex FFT Core Using Radix-8 Algorithm

The FPGA summary report and ASIC flow results are also given for the various FFT cores in the chapter five. A new architecture for 512-Point Complex FFT Core Using Radix-8 Algorithm has also been proposed in chapter five. The conclusions and future scope of work of this thesis have been described in chapter six.

Chapter 2

Fast Fourier Transform (FFT)

Introduction

Analysis and Calculating FFT

Cooley and Tukey Algorithm

Radix-4 FFT Algorithm

Split-Radix FFT Algorithm

Radix-8 FFT Algorithm

Butterfly Unit Implementation Results

Chapter 2

Fast Fourier Transform (FFT)

2.1 Introduction

Fast Fourier transform (FFT) [1] – [2] is an efficient algorithm to compute the discrete Fourier transform (DFT) and its inverse. There are many FFT algorithms involving a wide range of algorithm. A DFT decomposes a sequence of values into different frequencies components. This operation is useful in many fields but computing it direct computing from its definition is too slow to be practical. An FFT computation gives same result more quickly by computing a DFT of *N* points in the naive way, using the definition, takes N^2 complex multiplications and *N(N-1)* complex additions, while an FFT can compute the same result in only *N/2* log *N* complex multiplications and *N* log *N* complex additions. The difference in speed can be substantial for long data sets where data may be in the thousands or millions.

Many designs have been proposed for faster calculation of FFT. Various higher radix FFT algorithms like Radix-4, Radix-8, Radix-16, Radix-2^k, and Split-Radix FFT algorithms have been proposed for increasing the performance of FFT cores of small and large data sequences [5] – [13]. In many of the designs, pipelining and folding techniques have been used to increase throughput and speed [22] – [25]. Multiplexers, ROMs, or memory has been used in almost all the above proposed designs, thus making those designs inefficient in terms of area, power consumption, and speed. In this chapter, we will cover the various FFT algorithms like Radix-4, Split-Radix, and Radix-8 FFT algorithms and efficient ways to implement those in FPGA.

2.2 Analysis and Calculating FFT

The *N-point* DFT of a sequence $x(n)$ is given by

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk}$$

$$k = 0,1, \dots \dots, N-1 \qquad (1)$$

where $W_N^{nk} = e^{-j2\pi nk/N}$ is known as the twiddle factor.

Similarly, the formula for IDFT is given by

$$x(n) = 1/N \sum_{k=0}^{N-1} X(k) W_N^{-nk}$$

$n = 0,1, \ldots \ldots, N-1$

(2)

Since DFT and IDFT involve basically the same type of computations, all the efficient computational algorithms for the DFT applies as well to the efficient computation of the IDFT. We observe that direct computation of $X(k)$ involves N complex multiplications ($4N$ real multiplications) and N-1 complex additions ($4N$-2 real additions) for each value of k. Hence, to compute all the N values of the DFT requires N^2 complex multiplications and $N(N-1)$ complex additions. The direct computation of the DFT is basically inefficient primarily because it does not exploit the symmetry and periodicity properties of the twiddle factor W_N, which are used in computational efficient FFT algorithms. In particular, the properties are:

Symmetry property: $W_N^{k+N/2} = -W_N^k$

Periodic property: $W_N^{k+N} = W_N^k$

2.3 Cooley and Tukey Algorithm

The Cooley - Tukey method was popularized by a publication of J. W. Cooley and J. W. Tukey in 1965 [4], but it was later discovered by Heideman & Burrus in 1984 that those two authors had independently re-invented an algorithm known to Carl Friedrich Gauss around 1805 (and subsequently rediscovered several times in limited forms). The most common FFT uses the Cooley - Tukey algorithm. This method follows the algorithm is a divide and conquer algorithm that recursively breaks down a DFT of any composite size $N = N_1N_2$ into many smaller DFTs of sizes N_1 and N_2, along with O(N) complex multiplications by complex roots of unity called as twiddle factors.

The Cooley - Tukey algorithm is used to divide the transform into two pieces of size N/2 at each step, and is therefore limited to power of two sizes. These are called the radix-2 and mixed-radix cases, respectively (and other variants such as the split-radix FFT have their own names as well). Although the basic idea is recursive, most traditional implementations rearrange the algorithm to avoid explicit recursion. Also, because the Cooley–Tukey algorithm breaks the original DFT into smaller DFTs, it can be combined arbitrarily with any

other algorithm for the DFT, such as those described below. The basic equations for Radix-2 Decimation-in-Frequency (DIF) algorithm can be obtained by the above mentioned divide-and-conquer approach and are as follows.

$$X(2k) = \sum_{n=0}^{\frac{N}{2}-1} [x(n) + x\left(n + \frac{N}{2}\right)] W_{N/2}^{nk}$$

$$k = 0,1, \ldots\ldots, \frac{N}{2} - 1$$

(3)

$$X(2k+1) = \sum_{n=0}^{\frac{N}{2}-1} [x(n) - x\left(n + \frac{N}{2}\right)] W_N^n W_{N/2}^{nk}$$

$$k = 0,1, \ldots\ldots, \frac{N}{2} - 1$$

(4)

2.4 Radix-4 FFT Algorithm

When the number of samples *N* in the input sequence is a power of 4 (i.e., $N = 4^m$), it is more computationally efficient to employ a radix-4 FFT algorithm [1]. Let the number of samples in the input sequence is a product of two integers and is given by $N = L \times M$. The radix-4 decimation-in-frequency (DIF) FFT algorithm can be obtained by selecting *L=N/4* and *M=4*. The one-dimensional input sequence $x(n)$ can be stored as a two-dimensional array indexed by *l* and *m*, where $0 \leq l \leq L-1$and $0 \leq m \leq M-1$. Here, *l* is the row index and *m* is the column index. The output also can be indexed by *p* and *q* where $0 \leq p \leq L-1$ and $0 \leq q \leq M-1$. Hence for radix-4 case, *l*, *p*=0, 1,...., *N/4*-1 and *m*, *q*=0, 1, 2, 3. Taking column-wise storing of data samples that is *n= (N/4) m + l* and *k=4p + q*, the general expression of *N*-point DFT can be written as

$$X(p,q) = \sum_{l=0}^{\frac{N}{4}-1} G(l,q) W_{N/4}^{lp}$$

(5)

where

$$G(l,q) = W_N^{lq} F(l,q)$$

q=0, 1, 2, 3and l=0, 1,...., *N/4*-1

(6)

and

$$F(l,q) = \sum_{m=0}^{3} x(l,m) W_4^{mq}$$

q=0, 1, 2, 3and l=0, 1,...., *N/4*-1

(7)

From the above, it is clear that the *N*-point DFT is decimated into four *N/4*-point DFTs. Now breaking the original DFT expression into four smaller parts, we have

$$\begin{aligned}
X(k) &= \sum_{n=0}^{N-1} x(n) W_N^{kn} \\
&= \sum_{n=0}^{N/4-1} x(n) W_N^{kn} + \sum_{n=N/4}^{N/2-1} x(n) W_N^{kn} \\
&+ \sum_{n=N/2}^{3N/4-1} x(n) W_N^{kn} \\
&+ \sum_{n=3N/4}^{N-1} x(n) W_N^{kn} \\
&= \sum_{n=0}^{N/4-1} x(n) W_N^{kn} \\
&+ W_N^{Nk/4} \sum_{n=0}^{N/4-1} x(n+N/4) W_N^{kn} \\
&+ W_N^{kN/2} \sum_{n=0}^{N/4-1} x(n+N/2) W_N^{kn} + W_N^{3kN/4} \sum_{n=0}^{N/4-1} x(n+3N/4) W_N^{kn}
\end{aligned}$$

(8)

The phase factors can be calculated from its definition and are as follows

$W_N^{kN/4} = (-j)^k, W_N^{kN/2} = (-1)^k, W_N^{3kN/4} = (j)^k$

Substituting the twiddle factor values, we obtain

$$X(k) = \sum_{n=0}^{\frac{N}{4}-1} [x(n) + (-j)^k x\left(n + \frac{N}{4}\right) + (-1)^k x\left(n + \frac{N}{2}\right) + (j)^k x\left(n + \frac{3N}{4}\right)] W_N^{nk} \tag{9}$$

To convert the above into an *N*/4-point DFT, we subdivide $X(k)$ into four *N*/4-point subsequences, $X(4k), X(4k+1), X(4k+2), and\ X(4k+3), k$ =0, 1,...., *N*/4-1. So, the radix-4 DIF DFT are obtained as follows

$$X(4k) = \sum_{n=0}^{\frac{N}{4}-1} [x(n) + x\left(n + \frac{N}{4}\right) + x\left(n + \frac{N}{2}\right) + x\left(n + \frac{3N}{4}\right)] W_N^0 W_{N/4}^{kn}$$

$$X(4k+1) = \sum_{n=0}^{\frac{N}{4}-1} [x(n) - jx\left(n + \frac{N}{4}\right) - x\left(n + \frac{N}{2}\right) + jx\left(n + \frac{3N}{4}\right)] W_N^n W_{N/4}^{kn}$$

$$X(4k+2) = \sum_{n=0}^{\frac{N}{4}-1} [x(n) - x\left(n + \frac{N}{4}\right) + x\left(n + \frac{N}{2}\right) - x\left(n + \frac{3N}{4}\right)] W_N^{2n} W_{N/4}^{kn}$$

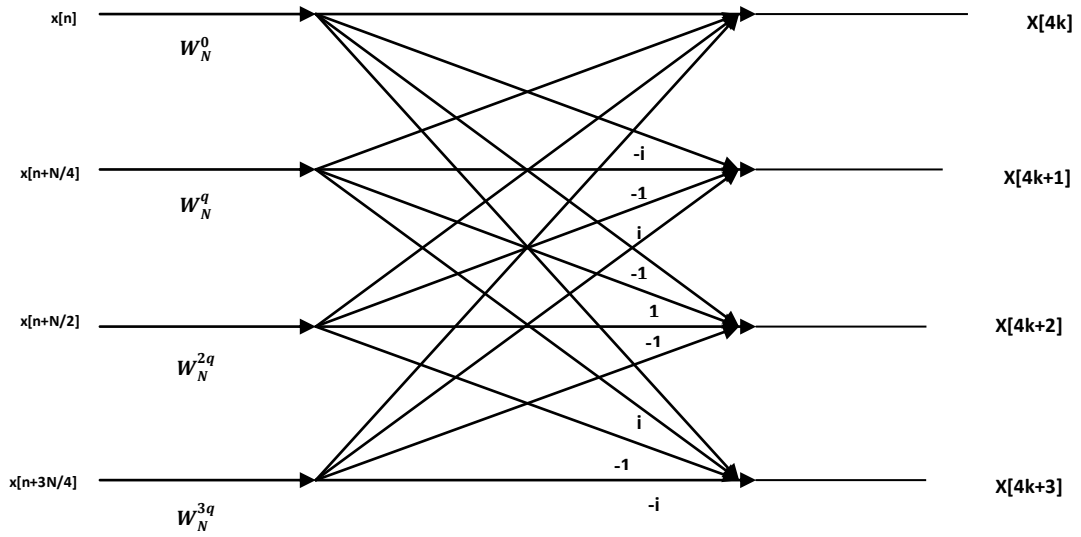

Figure 1. Radix-4 Butterfly Unit

$$X(4k+3)=\sum_{n=0}^{\frac{N}{4}-1}[x(n)+jx\left(n+\frac{N}{4}\right)-x\left(n+\frac{N}{2}\right)-jx\left(n+\frac{3N}{4}\right)]W_N^{3n}W_{N/4}^{kn}$$

(10)

2.5 Split-Radix FFT Algorithm

While calculating FFT using Radix-2 method, it can be concluded that the even-numbered points and the odd-numbered points are computed independently. This leads to the possibility of using different computational methods for different independent parts of the algorithm which will reduce computational complexity. Split-radix algorithm uses the above method by combining the simplicity of radix-2 algorithm and lesser computational complexity of radix-4 algorithm, achieving the lowest number of arithmetic operation count to compute DFT of power-of-two sizes *N*. Split-radix method recursively expresses DFT of length *N* in terms of one smaller DFT of length *N*/2 and two smaller DFTs of length *N*/4. Split-radix algorithm is only applicable when *N* is a multiple of 4, but we can combine this with other FFT algorithms.

The algorithm for the fast and less complexity computation of the DFT by Split-radix (SRFFT) was developed by Duhamel and Hollmann [30] for data sequences having a length *N* that is an integer power of 2. According to them, the even-numbered samples of the *N*-point DFT can be calculated by (3). Those even-numbered DFT points can be calculated without any additional multiplications. So, radix-2 algorithm is sufficient for the above calculation. The odd-numbered samples $X(2k+1)$ require an additional multiplication of W_N^n . To implement this, radix-4 algorithm is used for its lesser computational complexity. Using radix-4 algorithm for the odd-numbered samples of the *N*-point DFT, the following N/4-point DFT_s are obtained.

$$X(4k+1)=\sum_{n=0}^{\frac{N}{4}-1}[\left\{x(n)-x\left(n+\frac{N}{2}\right)\right\}-j\left\{x\left(n+\frac{N}{4}\right)-x\left(n+\frac{3N}{4}\right)\right\}]W_N^nW_{N/4}^{nk}$$

and

$$X(4k+3)=\sum_{n=0}^{\frac{N}{4}-1}[\left\{x(n)-x\left(n+\frac{N}{2}\right)\right\}+j\left\{x\left(n+\frac{N}{4}\right)-x\left(n+\frac{3N}{4}\right)\right\}]W_N^{3n}W_{N/4}^{nk}$$

$$k = 0,1, \ldots \ldots, \frac{N}{4} - 1$$

(11)

Hence, the *N*-point DFT now has been decomposed into one *N/2*-point DFT without phase factor and another two *N/4*-point DFT$_s$ with phase factor.

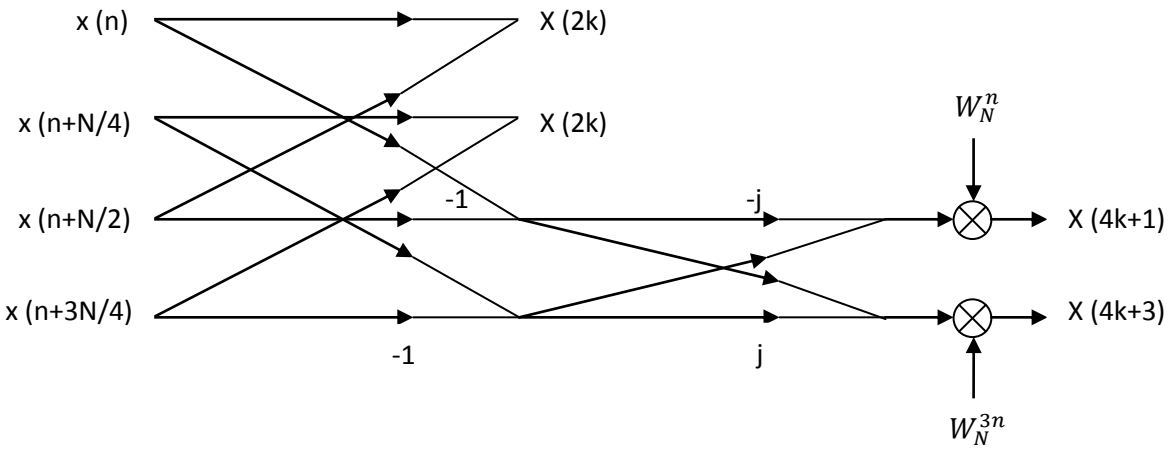

Figure 2. Split-Radix Butterfly Unit

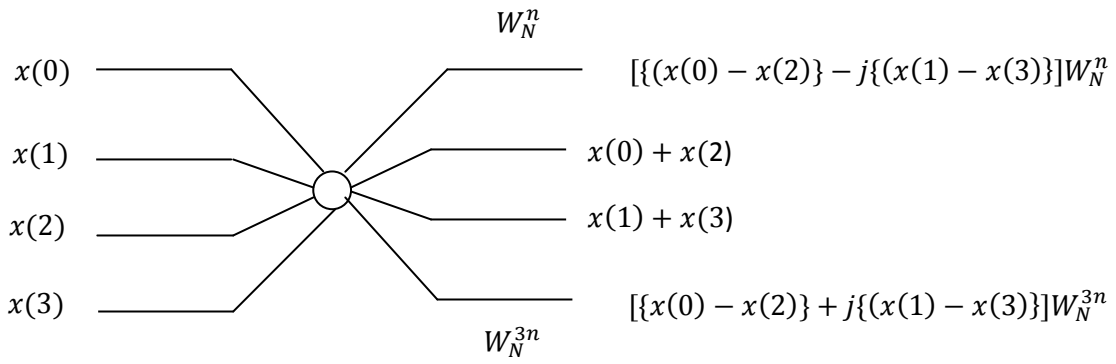

Figure 3. Split-Radix Butterfly with Permuted Outputs

2.6 Radix-8 FFT Algorithm

The number of multiplications and memory accesses dominate the power consumption of FFT computation. Its power consumption can be reduced significantly by using high-radix algorithms because the high radix algorithm can reduce the number of multiplications and also can reduce memory access times. In other words, higher radix algorithms reduce the computational complexity and computation time. For those reasons, higher radix algorithms

are best suitable for calculating FFT of large size. The radix-8 FFT algorithm [31] – [32] is as follows.

Let the number of samples in the input sequence is a product of two integers and is given by $N = L \times M$. The radix-8 decimation-in-frequency (DIF) FFT algorithm [1] can be obtained by selecting $L=N/8$ and $M=8$. The one-dimensional input sequence $x(n)$ can be stored as a two-dimensional array indexed by l and m, where $0 \leq l \leq L-1$and $0 \leq m \leq M-1$. Here, l is the row index and m is the column index. The output also can be indexed by p and q where $0 \leq p \leq L-1$ and $0 \leq q \leq M-1$. Hence for radix-8 case, l, p=0, 1,...., $N/8$-1 and m, q=0, 1, 2, 3, 4, 5, 6, 7. Taking column-wise storing of data samples that is $n= (N/8) m + l$ and $k=8p + q$, the general expression of N-point DFT can be written as

$$X(p,q) = \sum_{l=0}^{\frac{N}{8}-1} G(l,q) W_{N/8}^{lp} \tag{12}$$

where

$$G(l,q) = W_N^{lq} F(l,q)$$

q=0, 1, 2, 3, 4, 5, 6, 7 and l=0, 1,...., $N/8$-1

(13)

and

$$F(l,q) = \sum_{m=0}^{7} x(l,m) W_8^{mq}$$

q=0, 1, 2, 3, 4, 5, 6, 7 and l=0, 1,...., $N/8$-1

(14)

From the above, it is clear that the N-point DFT is decimated into eight $N/8$-point DFTs. We can write the above in matrix form as follows.

$$\begin{bmatrix} X(n) \\ X\left(n+\frac{N}{8}\right) \\ X\left(n+\frac{N}{4}\right) \\ X\left(n+\frac{3N}{8}\right) \\ X\left(n+\frac{N}{2}\right) \\ X\left(n+\frac{5N}{8}\right) \\ X\left(n+\frac{3N}{4}\right) \\ X(n+\frac{7N}{8}) \end{bmatrix} = [W_8^n]_{8\times 8} \begin{bmatrix} W_N^0 x(n) \\ W_N^q x\left(n+\frac{N}{8}\right) \\ W_N^{2q} x\left(n+\frac{N}{4}\right) \\ W_N^{3q} x\left(n+\frac{3N}{8}\right) \\ W_N^{4q} x\left(n+\frac{N}{2}\right) \\ W_N^{5q} x\left(n+\frac{5N}{8}\right) \\ W_N^{6q} x\left(n+\frac{3N}{4}\right) \\ W_N^{7q} x(n+\frac{7N}{8}) \end{bmatrix} \tag{15}$$

The mathematical expression for radix-8 butterfly element is as follows.

$$X(n) = \sum_{n=0}^{\frac{N}{8}} [[(x(n) + x\left(n+\frac{N}{2}\right)) + \left(x\left(n+\frac{N}{4}\right) + x\left(n+\frac{3N}{4}\right)\right)] + [(x\left(n+\frac{N}{8}\right) + x(n + \frac{5N}{8})) + (x\left(n+\frac{3N}{8}\right) + x(n+\frac{7N}{8}))]]W^{8nk}W^0$$

$$X(n+N/8) = \sum_{n=0}^{\frac{N}{8}} [[\left(x(n) - x\left(n+\frac{N}{2}\right)\right) - j\left(x\left(n+\frac{N}{4}\right) - x\left(n+\frac{3N}{4}\right)\right)] + W^{N/8}[(x\left(n + \frac{N}{8}\right) - x(n+\frac{5N}{8})) - j(x\left(n+\frac{3N}{8}\right) - x(n+\frac{7N}{8}))]]W^{8nk}W^q$$

$$X(n+N/4) = \sum_{n=0}^{\frac{N}{8}} [\left[\left(x(n) + x\left(n+\frac{N}{2}\right)\right) - \left(x\left(n+\frac{N}{4}\right) + x\left(n+\frac{3N}{4}\right)\right)\right] - j[(x\left(n+\frac{N}{8}\right) - x(n+\frac{5N}{8})) + j(x\left(n+\frac{3N}{8}\right) - x(n+\frac{7N}{8}))]]W^{8nk}W^{2q}$$

$$X(n+3N/8) = \sum_{n=0}^{\frac{N}{8}} [[\left(x(n) - x\left(n+\frac{N}{2}\right)\right) + j\left(x\left(n+\frac{N}{4}\right) - x\left(n+\frac{3N}{4}\right)\right)] + W^{3N/8}[(x\left(n+\frac{N}{8}\right) - x(n+\frac{5N}{8})) + j(x\left(n+\frac{3N}{8}\right) - x(n + \frac{7N}{8}))]]W^{8nk}W^{3q}$$

$$X(n+N/2)=\sum_{n=0}^{\frac{N}{8}}[\left[\left(x(n)+x\left(n+\frac{N}{2}\right)\right)+\left(x\left(n+\frac{N}{4}\right)+x\left(n+\frac{3N}{4}\right)\right)\right]-[(x\left(n+\frac{N}{8}\right)-x(n+\frac{5N}{8}))+(x\left(n+\frac{3N}{8}\right)+x(n+\frac{7N}{8}))]]W^{8nk}W^{4q}$$

$$X(n+5N/8)=\sum_{n=0}^{\frac{N}{8}}[[\left(x(n)-x\left(n+\frac{N}{2}\right)\right)-j\left(x\left(n+\frac{N}{4}\right)-x\left(n+\frac{3N}{4}\right)\right)]+W^{N/8}[(x\left(n+\frac{N}{8}\right)-x(n+\frac{5N}{8}))-j(x\left(n+\frac{3N}{8}\right)-x(n+\frac{7N}{8}))]]W^{8nk}W^{5q}$$

$$X(n+3N/4)=\sum_{n=0}^{\frac{N}{8}}[[\left(x(n)+x\left(n+\frac{N}{2}\right)\right)-\left(x\left(n+\frac{N}{4}\right)-x\left(n+\frac{3N}{4}\right)\right)]+j[(x\left(n+\frac{N}{8}\right)+x(n+\frac{5N}{8}))-(x\left(n+\frac{3N}{8}\right)+x(n+\frac{7N}{8}))]]W^{8nk}W^{6q}$$

$$X(n+7N/8)=\sum_{n=0}^{\frac{N}{8}}[\left[\left(x(n)-x\left(n+\frac{N}{2}\right)\right)+j\left(x\left(n+\frac{N}{4}\right)-x\left(n+\frac{3N}{4}\right)\right)\right]-W^{3N/8}[(x\left(n+\frac{N}{8}\right)-x(n+\frac{5N}{8}))+j(x\left(n+\frac{3N}{8}\right)-x(n+\frac{7N}{8}))]]W^{8nk}W^{7q}$$

(16)

However, these advantages of higher radix algorithm still did not be effectively utilized because the high-radix butterfly element will consume very large silicon area. The traditional hardware implementation of the butterfly element is direct mapping from the data flow to hardware. This approach is very simple but it is only suitable for the radix-2 or radix-4 butterfly element. For high radix butterfly element, the implementation of direct mapping will consume too large area. Here, the radix-8 FFT algorithm has been implemented using NEDA so that area will be reduced and throughput will still be high. The relation between inputs and outputs to implement the basic butterfly block of the radix-8 algorithm is given by

$$[X]_{8\times1}=[W_8^n]_{8\times8}\times[x]_{8\times1}$$

(17)

The final equation after using periodicity property of the twiddle matrix (that is $W_N^n = W_N^{mN+n}$, where m is an integer) is given by

$$\begin{bmatrix} X0Re + jX0Im \\ X1Re + jX1Im \\ X2Re + jX2Im \\ X3Re + jX3Im \\ X4Re + jX4Im \\ X5Re + jX5Im \\ X6Re + jX6Im \\ X7Re + jX7Im \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & W^0 & W^0 & W^0 & W^0 & W^0 \\ W^0 & W^1 & W^2 & W^3 & W^4 & W^5 & W^6 & W^7 \\ W^0 & W^2 & W^4 & W^6 & W^0 & W^2 & W^4 & W^6 \\ W^0 & W^3 & W^6 & W^1 & W^4 & W^7 & W^2 & W^5 \\ W^0 & W^4 & W^0 & W^4 & W^0 & W^4 & W^0 & W^4 \\ W^0 & W^5 & W^2 & W^7 & W^4 & W^1 & W^4 & W^3 \\ W^0 & W^6 & W^4 & W^2 & W^0 & W^6 & W^4 & W^2 \\ W^0 & W^7 & W^6 & W^5 & W^4 & W^3 & W^2 & W^1 \end{bmatrix} \times \begin{bmatrix} x0Re + jx0Im \\ x1Re + jx1Im \\ x2Re + jx2Im \\ x3Re + jx3Im \\ x4Re + jx4Im \\ x5Re + jx5Im \\ x6Re + jx6Im \\ x7Re + jx7Im \end{bmatrix}$$

(18)

The value of the twiddle factors used in the above matrix is given by: $W_8^0 = 1$, $W_8^1 = 0.707 - j0.707$, $W_8^2 = -j$, $W_8^3 = -0.707 - j0.707$, $W_8^4 = -1$, $W_8^5 = -0.707 + j0.707$, $W_8^6 = j$, and $W_8^7 = 0.707 + j0.707$. After multiplying the twiddle factors and simplifying, the expressions for real and imaginary parts of the outputs are given below.

$$X0Re = x0Re + x1Re + x2Re + x3Re + x4Re + x5Re + x6Re + x7Re$$

$$X0Im = j(x0Im + x1Im + x2Im + x3Im + x4Im + x5Im + x6Im + x7Im)$$

$$X1Re = x0Re + x2Im - x4Re - x6Im + 0.707(x1Re + x1Im - x3Re + x3Im - x5Re - x5Im + x7Re - x7Im)$$

$$X1Im = j[x0Im - x2Re - x4Im + x6Re + 0.707(-x1Re + x1Im - x3Re - x3Im + x5Re - x5Im + x7Re + x7Im)]$$

$$X2Re = x0Re + x1Im - x2Re - x3Im + x4Re + x5Im - x6Re - x7Im$$

$$X2Im = j(x0Im - x1Re - x2Im + x3Re + x4Im - x5Re - x6Im + x7Re)$$

$$X3Re = x0Re - x2Im - x4Re + x6Im + 0.707(-x1Re + x1Im + x3Re + x3Im + x5Re - x5Im - x7Re - x7Im)$$

$$X3Im = j[x0Im + x2Re - x4Im - x6Re + 0.707(-x1Re - x1Im - x3Re + x3Im + x5Re + x5Im + x7Re - x7Im)]$$

$$X4Re = x0Re - x1Re + x2Re - x3Re + x4Re - x5Re + x6Re - x7Re$$

$$X4Im = j(x0Im - x1Im + x2Im - x3Im + x4Im - x5Im + x6Im - x7Im)$$

$$X5Re = x0Re + x2Im - x4Re - x6Im + 0.707(-x1Re - x1Im + x3Re - x3Im + x5Re + x5Im - x7Re + x7Im)$$

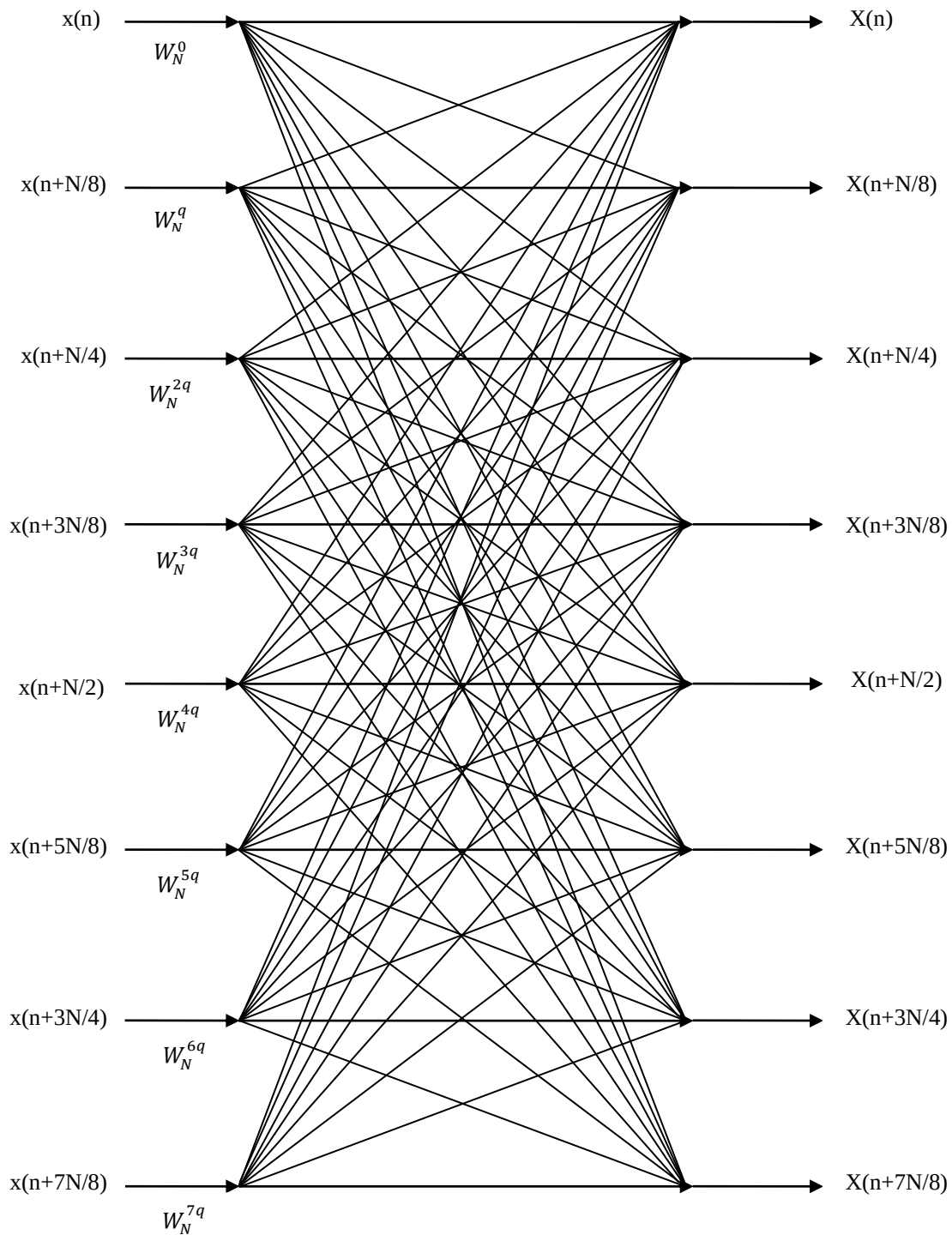

Figure 4. Radix-8 Butterfly Unit (the twiddle factors are given in (18))

$$X5Im = j[x0Im - x2Re - x4Im + x6Re + 0.707(x1Re - x1Im + x3Re + x3Im - x5Re + x5Im - x7Re - x7Im)]$$

$$X6Re = x0Re - x1Im - x2Re + x3Im + x4Re - x5Im - x6Re + x7Im$$

$$X6Im = j(x0Im + x1Re - x2Im - x3Re + x4Im + x5Re - x6Im - x7Re)$$

$$X7Re = x0Re - x2Im - x4Re + x6Im + 0.707(x1Re - x1Im - x3Re - x3Im - x5Re + x5Im + x7Re + x7Im)$$

$$X7Im = j[x0Im + x2Re - x4Im - x6Re + 0.707(x1Re + x1Im + x3Re - x3Im - x5Re - x5Im - x7Re + x7Im)]$$

(19)

The above equations have been implemented in VHDL and NEDA technique has been used for performing multiplications present in the above equations.

2.7 Butterfly Unit Implementation Results

The Radix-4, Split-Radix and Radix-8 butterflies have been implemented using Xilinx ISE on Virtex-5 FPGA. ASIC implementation of the butterfly structures in 0.18µm process technology using Synopsys DC for logic synthesis and Cadence SoC Encounter for physical design has also been carried out.

2.7.1 FPGA Device Utilization Summary

Table 1 shows the FPGA device utilization summary for the different butterfly units on XC5VLX330T-2FF1738.

Table 1. FPGA Device Utilization Summary of the Butterfly Units

FPGA device: XC5VLX330T-2FF1738	**Radix-4**	**Split-Radix**	**Radix-8**
Number of adders	15	15	163
Number of registers	128	128	504
Data path size	16 bits	16 bits	16 bits
Dynamic Power at maximum frequency	0.28710 W	0.03228 W	0.29813 W

2.7.2 ASIC Implementation Results

Table 2 shows the ASIC implementation of the proposed architectures in 0.18µm process technology using Synopsys DC for logic synthesis and Cadence SoC Encounter for physical design. The process technology that has been followed to carryout physical design of the proposed architectures is UMC 0.18µm mixed mode generic core.

Table 2. ASIC Implementation Results of the Butterfly Units Using Synopsys DC and Cadence SoC Encounter

ASIC implementation results using Synopsys DC	**Process technology: 0.18µm**		
	Radix-4	**Split-Radix**	**Radix-8**
Total cell area	22707.921776	23764.104183	177432.392842
Total dynamic power	1.4259 mW	2.0538 mW	15.8291 mW
Add-sub width	16 bits	16 bits	16 bits
Slack at 100 MHz	9.94 ns	6.62 ns	6.10 ns

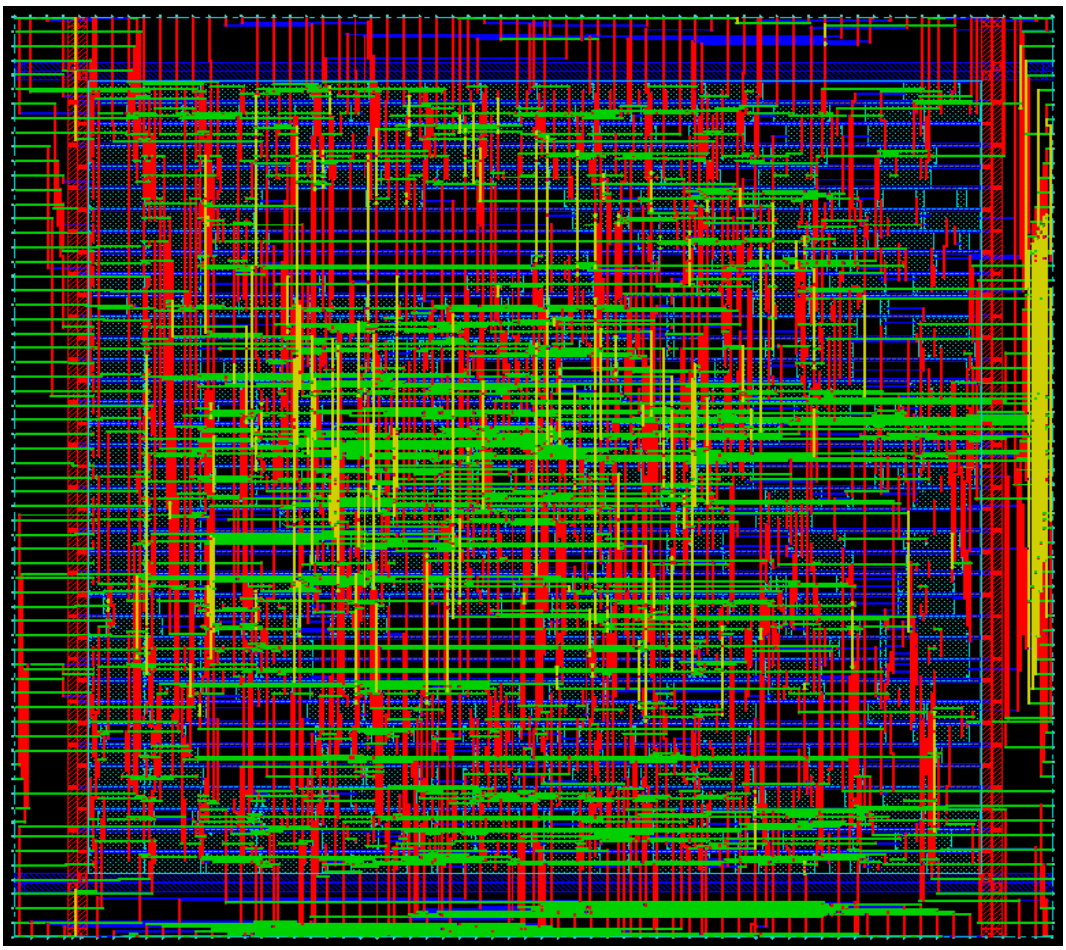

Figure 5. Physical Layout Radix-4 Butterfly

The physical design of proposed architectures has been made in such a way that the timing constraints are met after both placement as well as routing. The layouts for Radix-4, Split-radix, and Radix-8 are shown through figure 5 - 7 respectively.

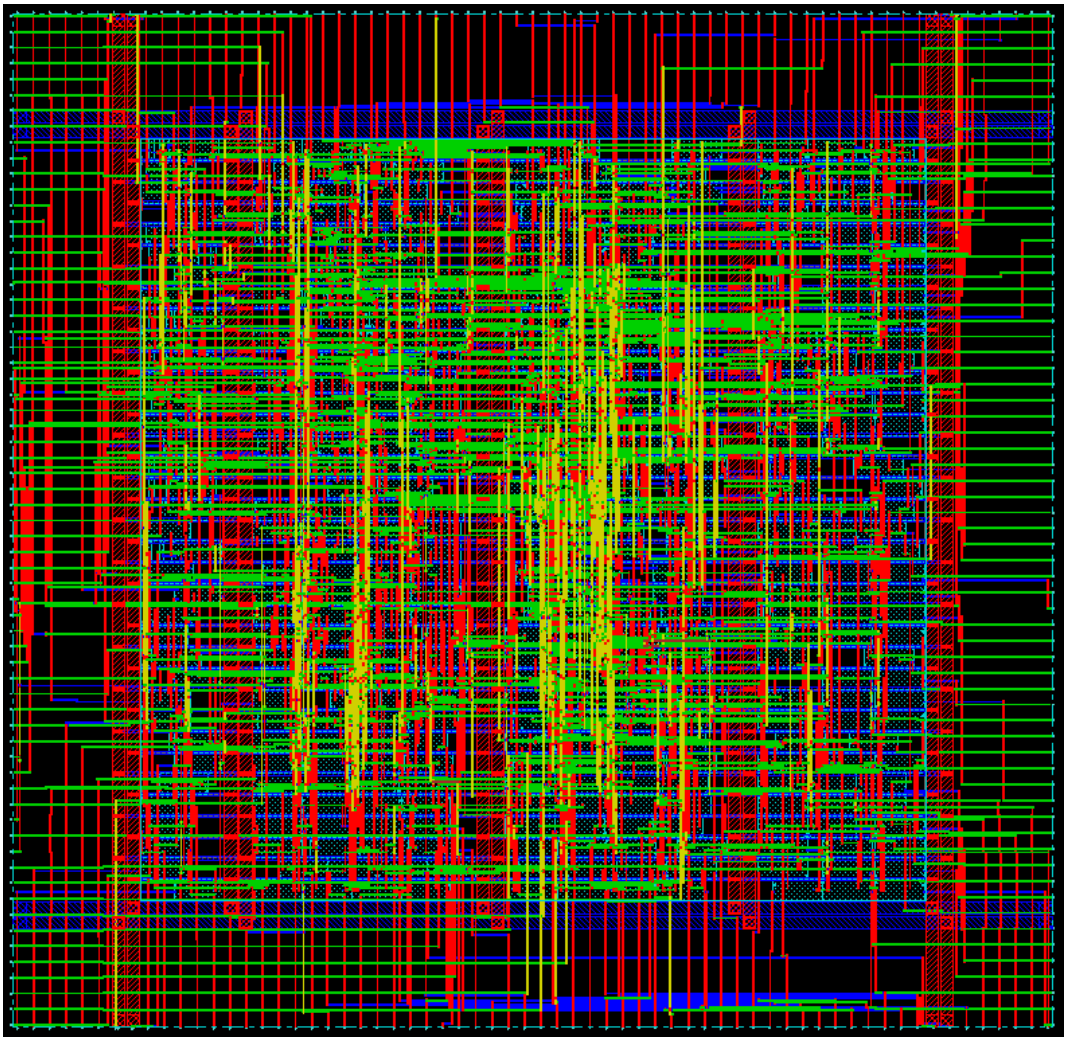

Figure 6. Physical Layout Split-Radix Butterfly

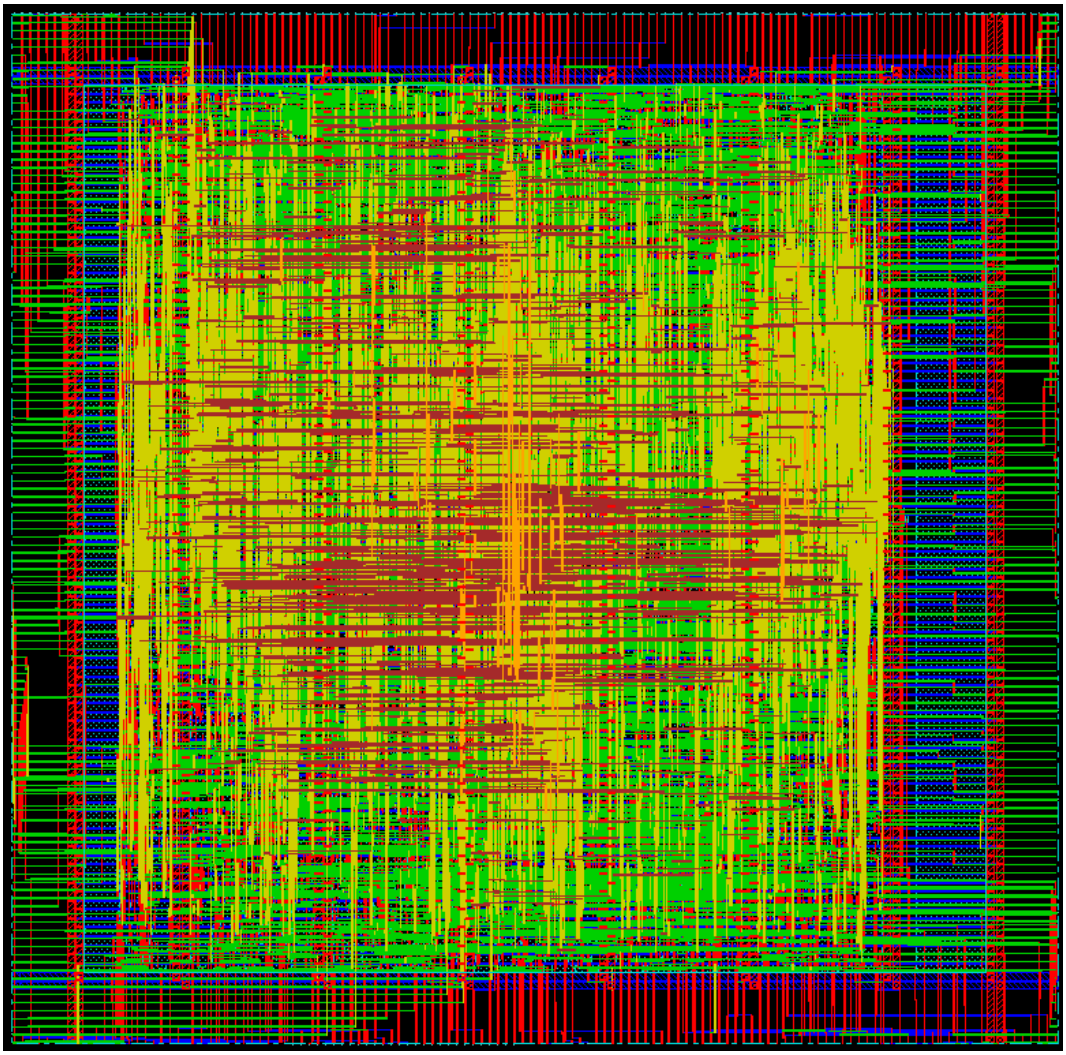

Figure 7. Physical Layout Radix-8 Butterfly

Chapter 3

New Distributed Arithmetic (NEDA)

Introduction

Description

Chapter 3

New Distributed Arithmetic (NEDA)

3.1 Introduction

Multiply and Accumulate (MAC) is one of the basic blocks used in many digital signal processing systems. The general structure of a MAC unit consists of a multiplier, an adder and a shifter. Elimination of multiplier in MAC unit can be made possible by using algorithms such as Distributed Arithmetic (DA) [17]. NEw Distributed Arithmetic (NEDA) [18] is similar to DA except that it has no ROM unit, which is used in the case of DA. Thus, NEDA eliminates the use of multipliers and ROM to carry out many computations such as Fast Fourier Transform (FFT), Discrete Cosine Transform (DCT), etc.

3.2 Description

The calculation of inner product of two sequences can be represented as

$$Z = \sum_{i=1}^{k} C_i X_i \tag{20}$$

Where C_i are constant fixed coefficients and X_i are varying inputs. Matrix representation of (20) may be given as

$$Z = [C_1 \quad C_2 \quad \cdots \quad C_k] \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_k \end{bmatrix} \tag{21}$$

Considering both C_i and X_i in 2's complement form, they can be expressed in the form

$$C_i = -C_i^M 2^M + \sum_{k=N}^{M-1} C_i^k 2^k \tag{22}$$

Where$C_i = 0\ or\ 1,\ k = N, N+1, \ldots, M$and C_i^M is the sign bit and C_i^N is the least significant bit. Substituting equation (22) in equation (21) results in the following matrix product which is modelled according to the required design of FFT.

$$Z = \begin{bmatrix} -2^0 & 2^{-1} & \cdots & 2^{-12} \end{bmatrix} \begin{bmatrix} C_1^0 & \cdots & C_k^0 \\ \vdots & \ddots & \vdots \\ C_1^{12} & \cdots & C_k^{12} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_k \end{bmatrix} \tag{23}$$

The matrix containing C_i^k is a sparse matrix, which means the values are either 1 or 0. The number of rows in C matrix defines the precision of fixed coefficients used. Equation (23) is rearranged as shown below.

$$Z = \begin{bmatrix} -2^0 & 2^{-1} & \cdots & 2^{-12} \end{bmatrix} \begin{bmatrix} W_0 \\ W_1 \\ \vdots \\ W_{12} \end{bmatrix} \tag{24}$$

Where $$\begin{bmatrix} W_0 \\ W_1 \\ \vdots \\ W_{12} \end{bmatrix} = \begin{bmatrix} C_1^0 & \cdots & C_k^0 \\ \vdots & \ddots & \vdots \\ C_1^{12} & \cdots & C_k^{12} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_k \end{bmatrix} \tag{25}$$

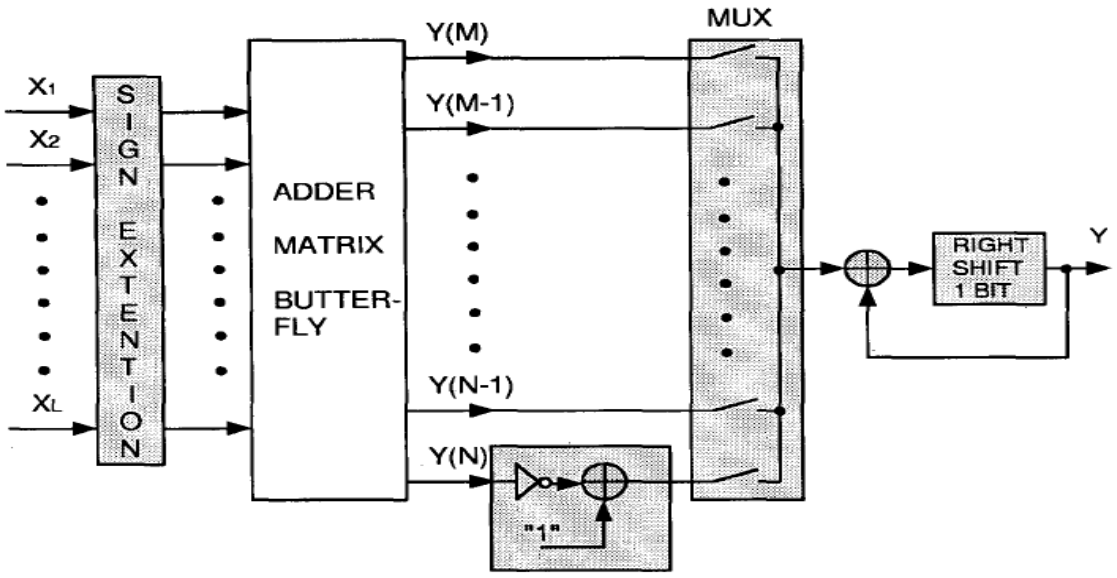

Figure 8. Architecture Implementation of NEDA [18]

Figure 8 shows the architecture implementing NEDA in which input signals are sign extended depending on the requirement and then fed into the adder matrix. The adder matrix is a butterfly structure whose number of output lines is determined by precision of the constant coefficients. The output Y in the shown figure takes as many clock cycles as the number of DA precision bits to get the correct value. To obtain Y in single clock cycle, the shown MUX and accumulator can be replaced by an adder compressor. In each row, the W matrix consists of sums of the inputs depending on the coefficient values. An example that shows the NEDA operations is discussed below. Consider to evaluate the value of equation (26).

$$Y = \begin{bmatrix} \cos\frac{\pi}{8} & \cos\frac{\pi}{4} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} \tag{26}$$

Equation (26) can be expressed in the form of equation (23) as shown in equation (27).

$$Y = \begin{bmatrix} -2^0 & 2^{-1} & \cdots & 2^{-12} \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 1 & 1 \\ 1 & 0 \\ 1 & 1 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \\ 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} \tag{27}$$

Equation (27) may be rewritten as

$$Y = [-2^0 \quad 2^{-1} \quad \cdots \quad 2^{-12}] \begin{bmatrix} 0 \\ X_1 + X_2 \\ X_1 \\ X_1 + X_2 \\ X_2 \\ X_1 \\ X_1 + X_2 \\ 0 \\ X_2 \\ X_1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{28}$$

Applying precise shifting, we rewrite equation (28) as

$$Y = [2^{-1} \quad 2^{-2} \quad 2^{-3} \quad 2^{-4} \quad 2^{-5} \quad 2^{-6} \quad 2^{-8} \quad 2^{-9}] \begin{bmatrix} X_1 + X_2 \\ X_1 \\ X_1 + X_2 \\ X_2 \\ X_1 \\ X_1 + X_2 \\ X_2 \\ X_1 \end{bmatrix} \tag{29}$$

Thus implementing (29) further reduces number of adders compared to implement (28). Multiplication with $2^{-i}, i \in Z^+$ can be realized with the help of arithmetic shifters. In (29), the first row of X matrix shifts right by 1 bit, second row by 2 bits and so on. More precisely, the shifts carried out are arithmetic right shifts. The output Y can be realized as a column matrix when we need the partial products. Thus, NEDA based architecture designs have less critical path compared to traditional MAC units without multipliers as well as memory.

Chapter 4

LabVIEW and CompactRIO FPGA

Introduction

The RIO Architecture

cRIO-9104

cRIO-9201

cRIO-9263

Chapter 4

LabVIEW and CompactRIO FPGA

4.1 Introduction

CompactRIO [33] combines a real-time embedded processor, a high-performance FPGA, and hot- swappable I/O modules mounted on the chassis. Each I/O module present on the chassis is connected directly to the FPGA. The above provides low-level customization of timing and I/O signal processing. The FPGA is connected to the real-time embedded processor via a high-speed duplex PCI bus. This provides open access to low-level hardware resources. This also represents a low-cost architecture. LabVIEW contains built-in data transfer mechanisms to pass data to the FPGA from the I/O modules and also to the embedded processor from the FPGA for real-time analysis, post data processing, data logging, or communication to a linked host computer. CompactRIO requires LabVIEW, Real-time processor, C-series I/O modules, chassis, CompactRIO device driver and FPGA tool kits to function as a complete development suite.

4.2 The RIO architecture

The CompactRIO embedded system features an industrial 400 MHz Freescale MPC5200 processor. It deterministically executes LabVIEW Real-Time applications on the reliable Wind River VxWorks real-time operating system. Built-in functions of LabVIEW transfer data between the FPGA and the real-time embedded processor within the CompactRIO embedded system. Existing C/C++ code can also be integrated with LabVIEW Real-Time code to save on development time. A variety of swappable I/O types are available including voltage, RTD, current, thermocouple, accelerometer, and strain gauge inputs. There are analog I/O modules, digital I/O modules, counter/timers, high voltage/current relays, and pulse generation units with a wide range current and voltage rating. Sensors and actuators can be wired directly with C series modules as the modules contain built-in signal conditioning for extended voltage ranges for industrial signal types. The embedded FPGA is a high-performance, ultra ruggedness reconfigurable chip that can be programmed with LabVIEW FPGA tools. Historically, FPGA designers were forced to learn and use complex design languages such as Verilog or VHDL to program the FPGAs. Now, the problem can be solved by using LabVIEW tools to program for self-customized FPGAs. One can implement custom timing, synchronization, triggering, control, and signal processing for your analog and digital

I/O using the FPGA hardware embedded in CompactRIO. The RIO architecture has been shown in figure 9.

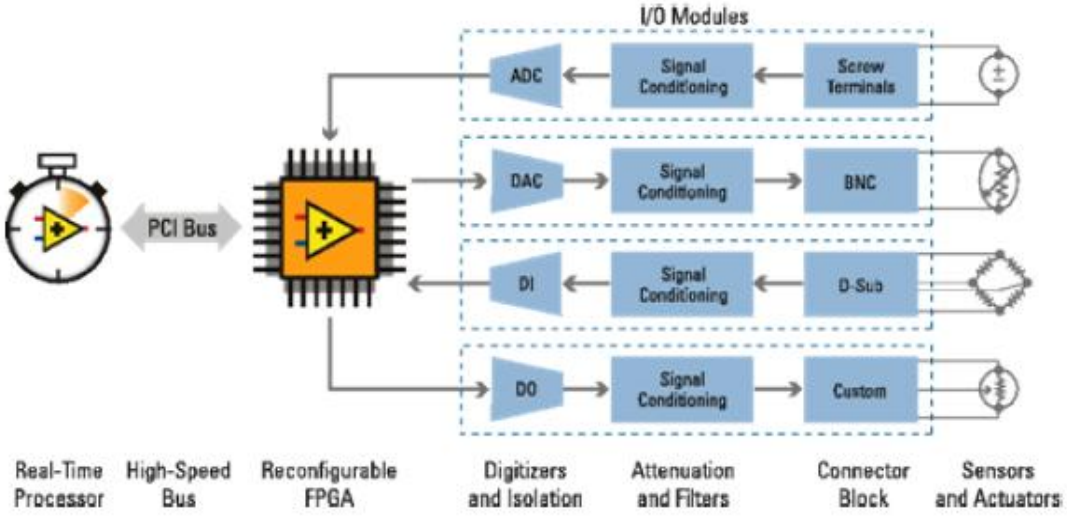

Figure 9. The Rio Architecture [33]

4.3 cRIO-9104

The NI-9104 [34] is a real-time intelligent embedded processor with 400 MHz of operating frequency, 128 MB of DRAM, 2 GB of memory in which data transfer can takes place at a speed of 10 Mbps. The cRIO-9104 is shown in figure 10.

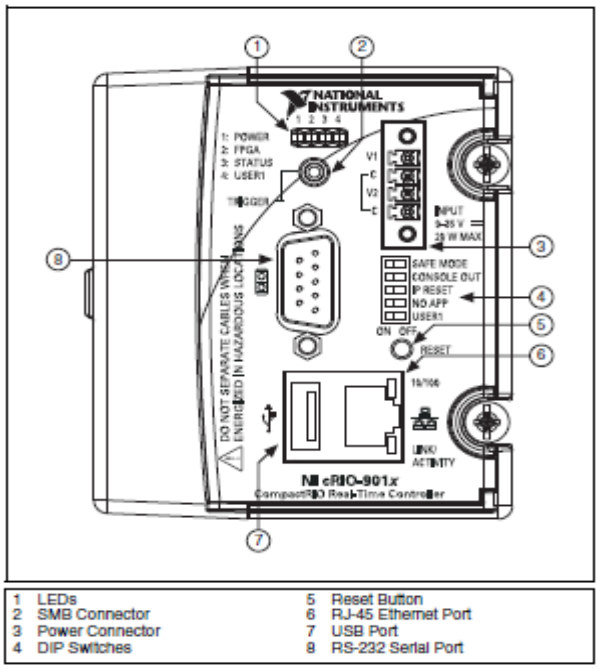

Figure 10. cRIO-9104 [34]

The important specifications of cRIO-9104 are given in table 3.

Table 3. Specifications of cRIO-9104, reduced from [34]

Product name	Crio-9104
Network interface	10BaseT and 100BaseTX Ethernet
Compatibility	IEEE 802.3
Communication rates	10 Mbps, 100 Mbps, auto-negotiated
Maximum cabling distance	100 m/segment
Logic high	3.3 V
Logic low	0 V
Maximum input level	-500 mV
Minimum input level	5.5 V
Non volatile memory	2 GB
DRAM	128 MB
Power consumption (controller only)	6 W
Minimum operating temperature	-40 ºC
Maximum operating temperature	70 ºC

4.4 cRIO-9201

The NI-9201 [35] is an 8-channel, 12-bit analog input module which uses successive approximation register (SAR) of analog to digital converter (ADC). The cRIO-9201 has a 10-terminal, and detachable screw-terminal connector that provides connections for eight analog input channels. Each channel has a terminal, AI, to which you can connect a voltage signal. The cRIO-9201 also has a common terminal (COM) which is internally connected to the isolated ground reference of the module. The cRIO-9201 channels are isolated from other modules in the CompactRIO system as shown in the figure 11. The module protects each channel from overvoltage. The input signals are scanned, buffered, conditioned, and are then sampled by a single 12-bit ADC. Some important specifications of Crio-9201 are given in table 4.

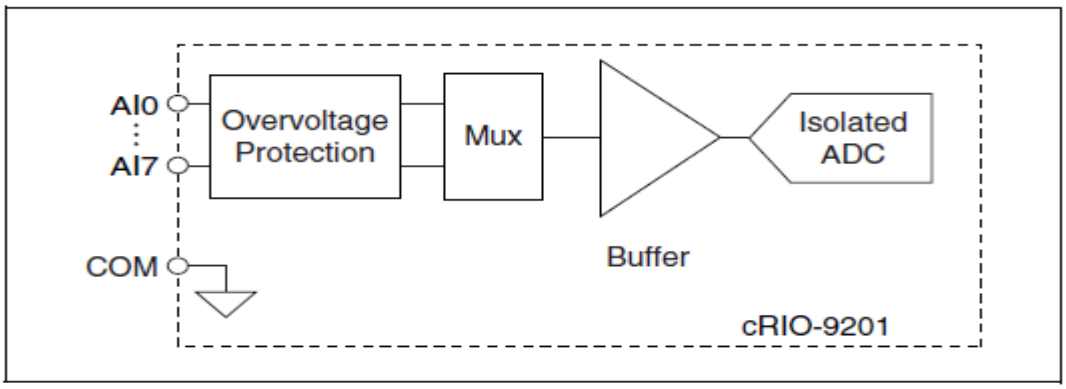

Figure 11. Input Circuitry for One Channel of cRIO-9201 [35]

Table 4. Specifications of cRIO-9201, reduced from [35]

Product name	Crio-9201
Number of channels	8
ADC type	Successive approximation register (SAR)
ADC resolution	12 bits
Typical operating voltage	±10.53 V
Sampling rate (aggregate)	500 kS/s
Input bandwidth (-3dB)	690 kHz
Input resistance	1 MΩ
Input capacitance	5 pF
DNL	-0.9 TO 1.5 LSBs max
INL	±1.5 LSBs max
Minimum operating temperature	-40 ºC
Maximum operating temperature	70 ºC
Maximum power consumption in active mode	550 mW

4.5 cRIO-9263

The NI-9263 [36] is a 4-channel, ±10V, 16-bit analog output module that works as a digital to analog converter (DAC). The cRIO-9263 has a 10-terminal, and detachable screw-terminal connector that provides connections for four analog output channels. Each channel has a terminal to which you can connect the positive lead of a voltage signal, AO. The cRIO-9263

also has common terminals (COM) that are internally connected to the isolated ground reference of the module. The cRIO-9263 channels share a common ground that is isolated from the other modules in the CompactRIO system as shown in the figure 12. Each channel has a digital-to-analog converter (DAC) that produces a voltage signal. One must write binary values to the analog output channels. Some important specifications of Crio-9263 are given in table 5.

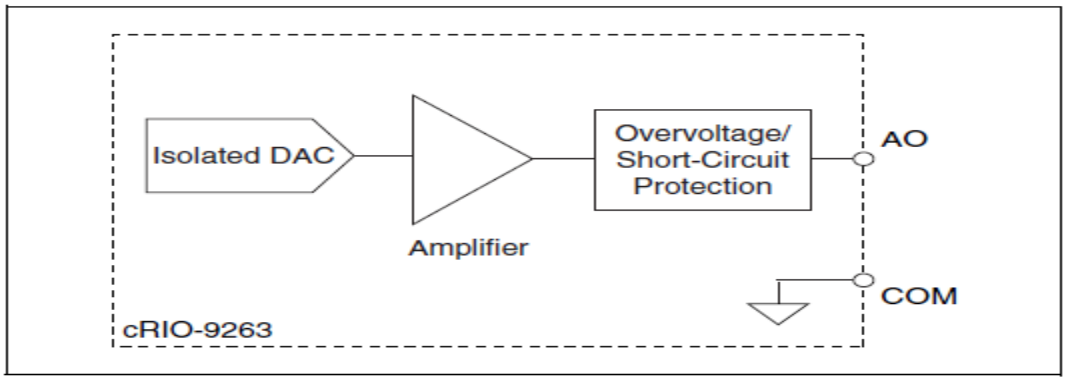

Figure 12. Output Circuitry for One Channel of cRIO-9263 [36]

Table 5. Specifications of cRIO-9263, reduced from [36]

Product name	Crio-9263
Number of channels	4
DAC type	String
DAC resolution	16 bits
Nominal operating voltage	±10.7 V
Update rate	100 kS/s
Output impedance	0.1 Ω
Slew rate	4 V/µs
DNL	-1 TO 2 LSBs max
INL (endpoint)	130 LSBs max
Minimum operating temperature	-40 ºC
Maximum operating temperature	70 ºC
Maximum power consumption in active mode	625 mW

Chapter 5

Different Fast Fourier Transform (FFT) Cores

Introduction

Real-time implementation of FFT using CompactRIO

32-Point Complex FFT Core Using Split-Radix Algorithm

64-Point Complex FFT Core Using Radix-4 Algorithm

512-Point Complex FFT Core Using Radix-8 Algorithm

Chapter 5

Different Fast Fourier Transform (FFT) Cores

5.1 Introduction

Using the different FFT algorithms as discussed in chapter 2, various complex FFT cores have been proposed. While the main idea is to make the FFT cores multiplier-less by using NEDA technique, still folding and pipelining techniques has also been used to increase the performance of the designs. First the FFT has been implemented using CompactRIO and LabVIEW for real-time signals. In the next sections, three different FFT cores have been described and these are:

- 32-Point Complex FFT Core Using Split-Radix Algorithm
- 64-Point Complex FFT Core Using Radix-4 Algorithm
- 64-Point Complex FFT Core Using Radix-8 Algorithm

The architectures have been implemented on both FPGA and ASIC. The proposed designs also have been compared with other existing designs. Finally, the architecture for a 512-Point Complex FFT Core using radix-8 algorithm has been described.

5.2 Real-time implementation of FFT using CompactRIO

5.2.1 Proposed Architecture

The idea is to find 256-point FFT of an analog signal given from the real world and finding the power spectrum of the input signal in a continuous manner. The proposed implementation is divided into two parts: writing the code in FPGA mode and controlling it through HOST [29]. The block diagram for the proposed architecture has been shown in figure 13. Here NI-9104 has been taken which is an intelligent real-time embedded controller for CompactRIO. An 8-slot chassis has been taken on which NI-9201 and NI-9263 are mounted. Prior to the final implementation a thorough testing of the listed hardware has been conducted by making a simple AI/AO loop with a scaling option for sine and square wave as input. Signal is given to NI-9201(MOD-3 AI) from function generator and is scaled and given to NI-9263(MOD-6 AO) from which the output is verified on a Cathode Ray Oscilloscope (CRO). The result confirms that the hardware works fine up to 50 kHz, 5V input signal with a scaling factor from 0.25 to 3 depending on the amplitude of the input signal.

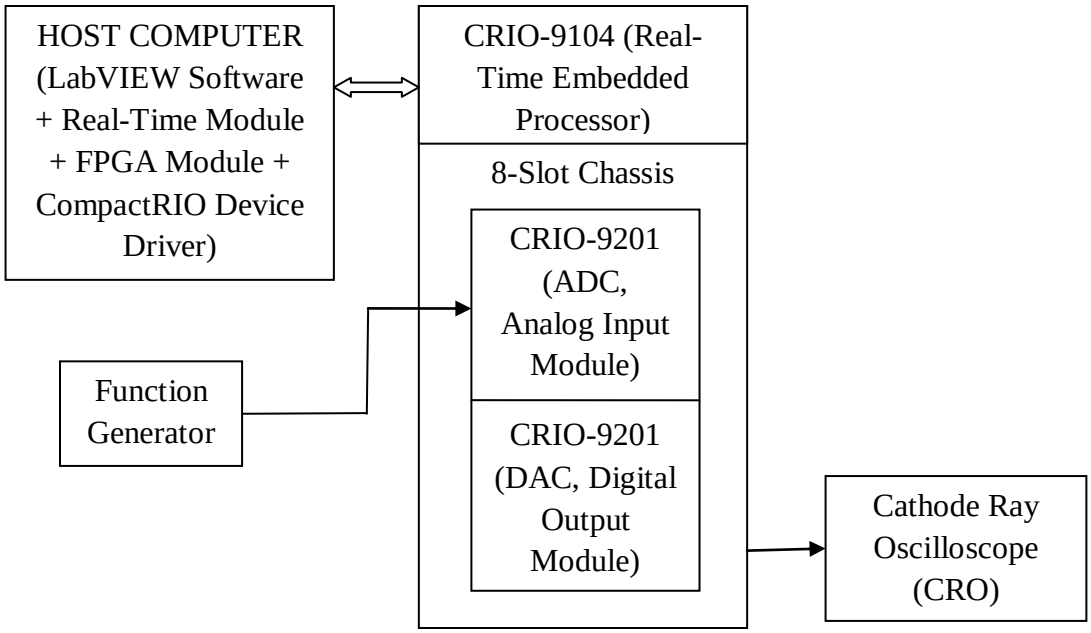

Figure 13. Block Diagram of the Proposed Implementation in LabVIEW

5.2.2 FPGA VI

Figure 14 shows the FPGA VI for finding the FFT of a real-time signal and storing its value in memory and using DMA-FIFO [27], [29] for the stored data to be transferred to the HOST. Signal generation VIs are placed outside the case structure in order to avoid duplicating logic by having the same functions in multiple cases. The signal is given to NI-9201 (MOD-3 AI). The digital output of NI-9201(ADC) is given to the FFT express VI which calculates the 256-point FFT and gives real and imaginary values. Those values are stored in memory separately. The memory used serves as a circular buffer that is continually overwritten with the latest FFT results. This is useful in situations where you don't need to capture all FFT results. The loop in the right side of flat sequence ensures that loop rate does not exceed the sample rate (ticks). Use the Flat Sequence structure to ensure that a sub diagram executes before or after another sub diagram. When all data values wired to a frame are available frames in a Flat Sequence structure starts executing from left to right. As the frame finishes executing, the data leaves each frame simultaneously. This shows the dependency of input and output of two consecutive frames. You do not need to use sequence locals to pass data from frame to frame in the Flat Sequence structure. The lower loop guarantees that FFT frames will be delivered coherently to the host in 256-element frames by checking the DMA FIFOs status before writing to them, so that the FPGA DMA buffer never overflows. The host VI can be arbitrarily slow and still get frames with the correct data order. There is never

a guarantee, however, that all data from the circular buffer makes it into the DMA FIFO before being overwritten. The DMA FIFO buffers may be set to a relatively small size in order to reduce the latency in display on the host. Large buffers are not necessary here since we are willing to drop frames in this display-only scheme.

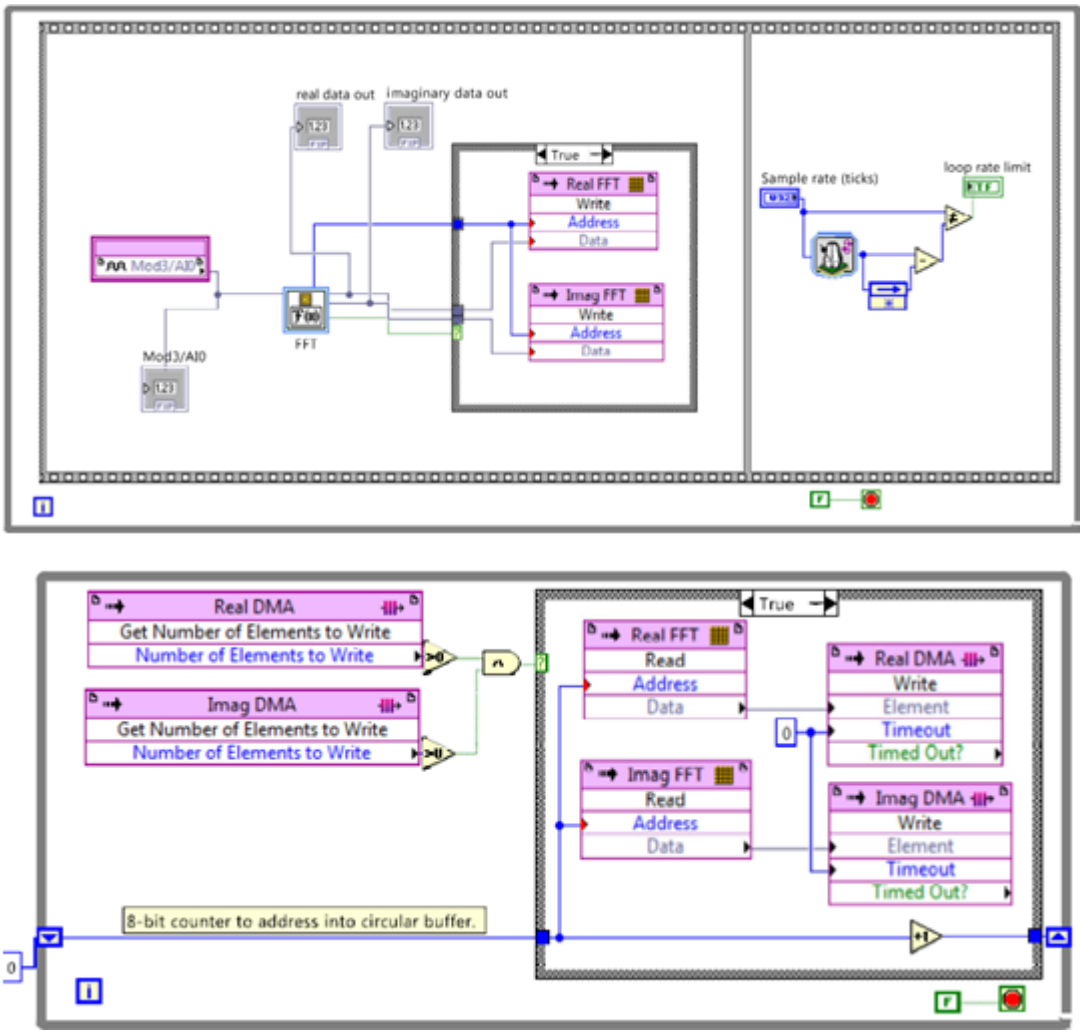

Figure 14. FPGA VI for the Proposed Implementation in LabVIEW

The advantage of DMA is that the host computer processor can perform calculations while the FPGA target transfers data to the host computer memory through bus mastering. FPGA targets which support DMA-FIFOs can master the PCI bus to directly access memory on the host computer without involving the host computer processor. A DMA-FIFO allocates memory on both the host computer and the FPGA target, but yet acts as a single FIFO. The FPGA VI writes to the DMA- FIFO one element at a particular instant of time with the write method of the FIFO method node or reads from the FIFO one element at a particular instant of time with the read method. While invoking, the host VI reads from or writes to the FIFO

one or more elements at a time. A DMA Engine is used by LabVIEW to transfer DMA-FIFO data between the FPGA and the host computer. The DMA Engine includes hardware logic and driver software. When the DMA Engine runs, it automatically transfers data between the DMA-FIFO memory on the FPGA and the DMA-FIFO memory on the host computer so they act as one FIFO array.

5.2.3 HOST VI

The HOST VI which uses the FPGA code is shown in figure 15. The values stored in memory are now transferred to the HOST through DMA-FIFO. The open FPGA reference opens a reference to the FPGA VI without running it, in order to avoid generating data before DMA is configured. Reset the VI to guarantee that FIFOs are in a known state on the target. Here the use of small host buffers is to minimize latency for signal changes to show up in the display of HOST VI.

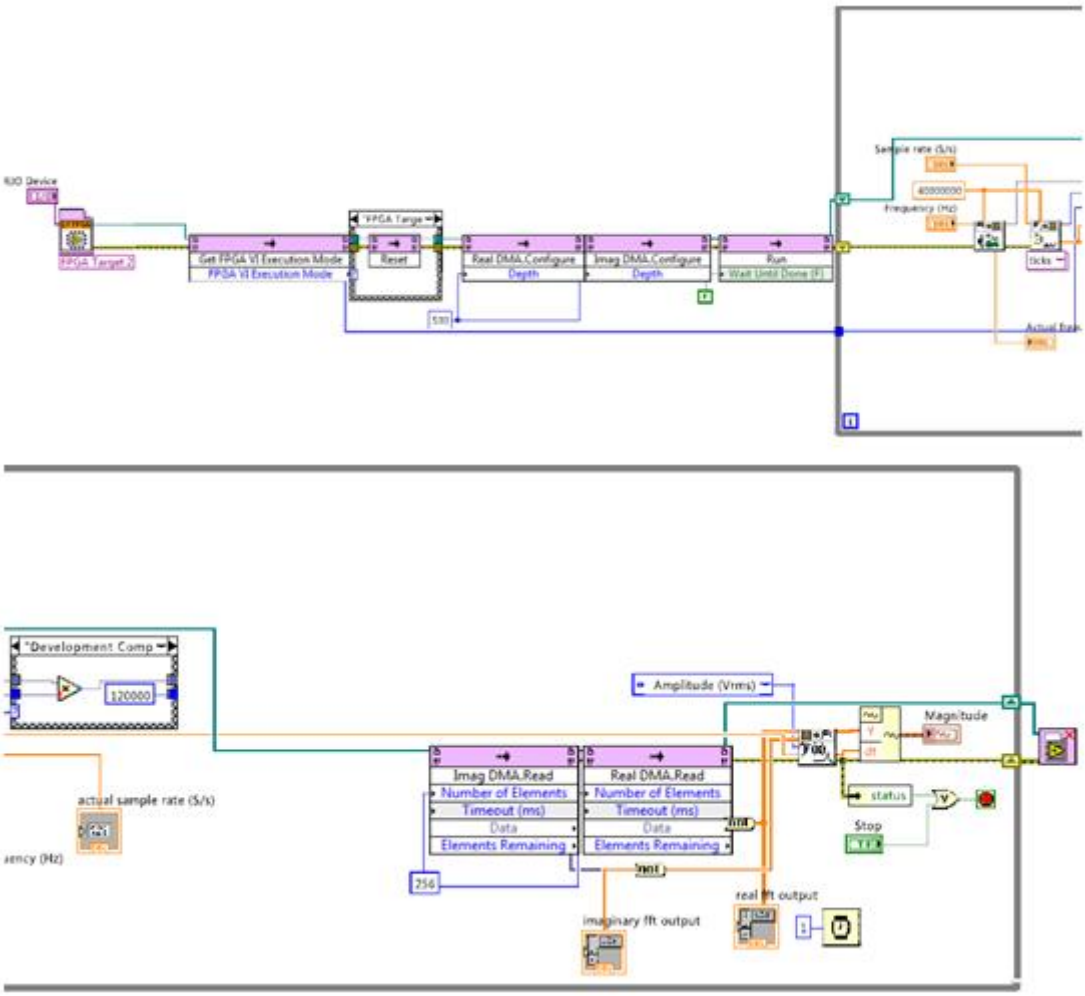

Figure 15. HOST VI for the Proposed Implementation in LabVIEW

The sample rate can be given manually or it can be determined by the loop by using different scaling VIs present in LabVIEW. The frequencies can also be seen or can be given manually. Since the FPGA VI prevents buffer overflows, the output should always be exactly one frame, starting with the DC bin. The FFT to spectrum VI finally gives the power spectrum of the input signal from the calculated FFT values. There are two memories created to store real and imaginary values of the FFT and are named as real FFT and imaginary FFT respectively. Likewise two DMA-FIFOs named as real DMA and imaginary DMA are being used to transport the values stored in the real and imaginary memory to the host respectively. Prior to the main loop in the HOST VI, real and imaginary DMA has been configured so that the main loop will run after the FIFOs get 500 values from FPGA VI, out of which 256 will be used at a time. This ensures that there will be continuous execution of the main loop in HOST VI without any data insufficiency. The values are converted to DBL (double precision float) from FXP (fixed point). The time delay module ensures clear vision of change of outputs from one frame to another. Finally the Close FPGA VI reference closes the reference to the FPGA.

5.2.4 Results

The FFT and power spectrum has been calculated for a 1 kHz, 10 kHz square wave with amplitude of 1V and the results are shown through figure 16 and 17 respectively. It is known in that the power spectrum of a square wave at a particular frequency (f_m) contains only odd harmonics. The amplitude of the odd harmonics is given by $2/(\pi \times n)$ where π=3.142 and n is the odd harmonic number. The first harmonic presents at that frequency with the power nearly equal to the input signal's amplitude value. The third harmonic presents at $3\times f_m$ with a power nearly equal to 0.21V, the fifth harmonic has a value around 0.12V present at a $5\times f_m$ for a 1V input signal and so on.

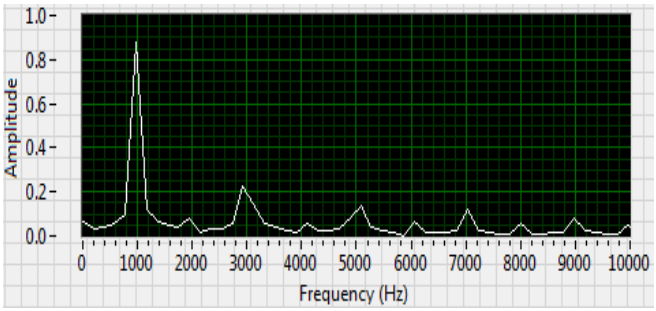

Figure 16. Observed Power Spectrum for 1 kHz Square Wave

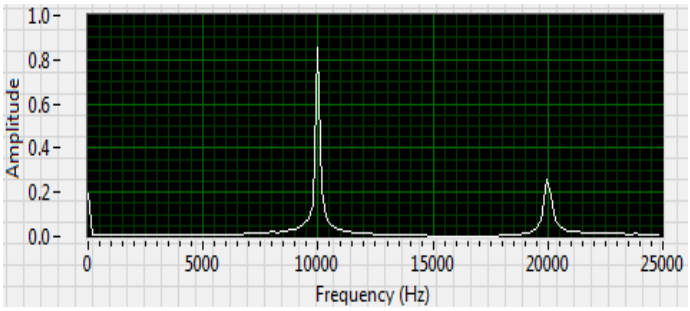

Figure 17. Observed Power Spectrum for 10 kHz Square Wave

5.2.5 Device Utilization Summary

The proposed implementation of 256-point FFT and its power spectrum using CompactRIO has been allocated only 3077 slices(21%), 2489 slice registers(8.7%), 4651 slice LUTs(16.2%) as shown in table 6. It can be operated with a maximum frequency of 52.42 MHz for on-board clock and 209.69 MHz for non-diagram components.

Table 6. Final Synthesis Report of the Proposed FFT Implementation Using CompactRIO

Device Utilization	Used	Total	Percent (%)
Total Slices	3077	14336	21.5
Slice Registers	2489	28672	8.7
Slice LUTs	4651	28672	16.2
Block RAMs	16	96	16.7

5.3 32-Point Complex FFT Core Using Split-Radix Algorithm

5.3.1 Proposed Architecture – I

In the proposed architecture – I, 32 complex inputs have been taken with a precession of 16 bits, in parallel. The number of stages to calculate the final output is 5. The inputs are taken in normal order and the outputs are in bit-reversal order. The even-numbered samples have been implemented by radix-2 FFT algorithm and the odd-numbered samples have been implemented using radix-4 FFT algorithm. The twiddle factor multiplications have been implemented using NEDA technique. The proposed architecture – I is shown in figure 18. In stage-I, eight radix-4 butterfly modules have been used. The inputs to each radix-4 butterfly present in stage-I are $x(n), x\left(n+\frac{N}{4}\right), x\left(n+\frac{N}{2}\right), x(n+\frac{3N}{4})$ where $0 \leq n \leq \frac{N}{4}-1$

respectively. The first output of each split-radix butterfly present in stage-I are represented by $S1(0), S1(1), \ldots\ldots, S1(7)$ respectively. The second output of each split-radix butterfly of stage-I are represented by $S1(8), S1(9), \ldots\ldots, S1(15)$ respectively. Similarly the third and fourth output of each split-radix butterfly of stage-I are represented as $S1(16), S1(17), \ldots, S1(23)$ and $S1(24), S1(26), \ldots, S1(31)$ respectively.

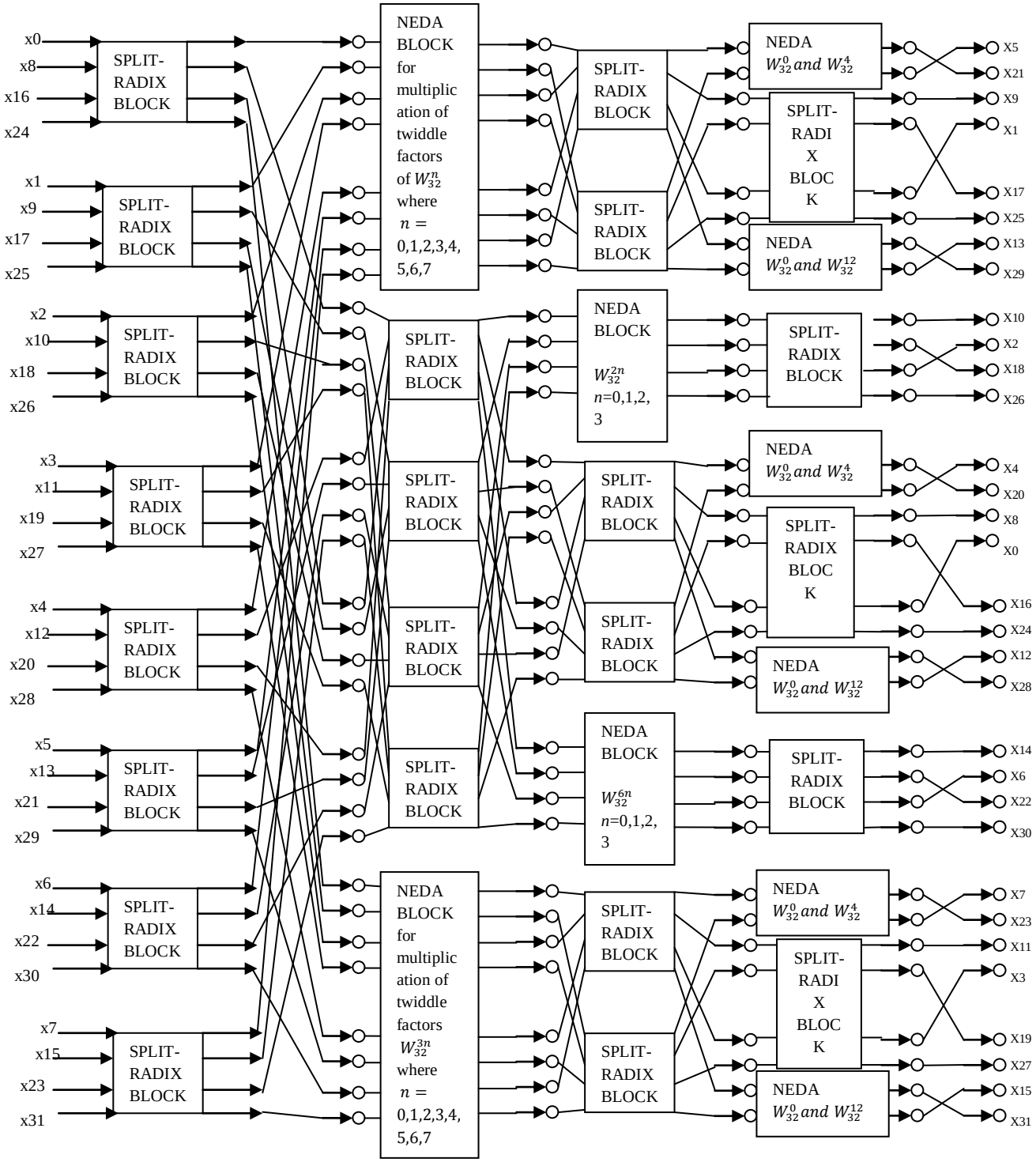

Figure 18. Proposed Architecture – I of 32-Point Split-Radix FFT

In stage-II, the samples $S1(0), S1(1), \ldots\ldots, S1(7)$ are multiplied by twiddle factor of W_N^n and the samples $S1(24), S1(26), \ldots\ldots, S1(31)$ are multiplied by twiddle factor of W_N^{3n} where N=32 and $0 \leq n \leq \frac{N}{4} - 1$ respectively. Those inner product calculations have been done by NEDA technique to achieve a multiplier-less architecture. The rest of stage-I samples are fed to four split-radix butterfly units and the outputs are given to stage-III. In stage-III, the samples $\{S2(0), S2(1), \ldots\ldots, S2(7)\}$, $\{S2(12), S2(13), \ldots\ldots, S2(19)\}$, $\{S2(24), S2(25), \ldots, S2(31)\}$ are fed to six split-radix butterfly units and the outputs are given to stage-IV respectively. The remaining samples of stage-III are multiplied by twiddle factor of W_N^{2n} and W_N^{6n} where N=32 and $0 \leq n \leq \frac{N}{8} - 1$ respectfully.

In stage-IV, five more split-radix butterfly units have been used and the inputs and outputs of those are clearly shown in figure. The twiddle factor that is to be multiplied in stage-IV whenever required is W_N^{4n} and W_N^{12n} where N=32 and $0 \leq n \leq \frac{N}{16} - 1$. The final stage (stage-V) uses only radix-2 butterfly units whenever required. The twiddle factor to be multiplied in stage-V is W_N^0 since $0 \leq n \leq \frac{N}{32} - 1$ that is n=0. The NEDA technique has been used here whenever there is a need for the calculation of inner products. We got the final output at the end of stage-V.

5.3.2 Proposed Architecture – II

The draw-back of the proposed architecture – I lies in its huge number of input-output pins, which makes the design less implementable both on FPGAs as well as an ASIC. To overcome the above draw-back, an intelligent way of implementing the split-radix FFT is done through folding.

The proposed architecture – II, shown in figure 19, takes 4 inputs at a time which sums up to 8 clock cycles to read all the 32 inputs. The outputs of the first stage split-radix block are stored in registers for every clock cycle and this process continues till all 32 outputs are stored. Later, the stored outputs are processed for second stage computations which consist of either NEDA blocks or split-radix blocks. The outputs of second stage split-radix blocks are stored in 16 registers for further processing. The outputs of second stage NEDA blocks and some outputs of second stage split-radix blocks are given to third stage split-radix blocks. The remaining outputs of second stage split-radix blocks are given to NEDA blocks of third stage. Some outputs of third stage split-radix blocks are given to fourth stage NEDA

blocks. The remaining outputs of third stage split-radix blocks along with third stage NEDA blocks are given to fourth stage split-radix blocks. The outputs of fourth stage NEDA blocks and some outputs of fourth stage split-radix blocks are fed to fifth stage radix-2 blocks. Rest of the outputs of fourth stage split-radix blocks are directly mapped to outputs.

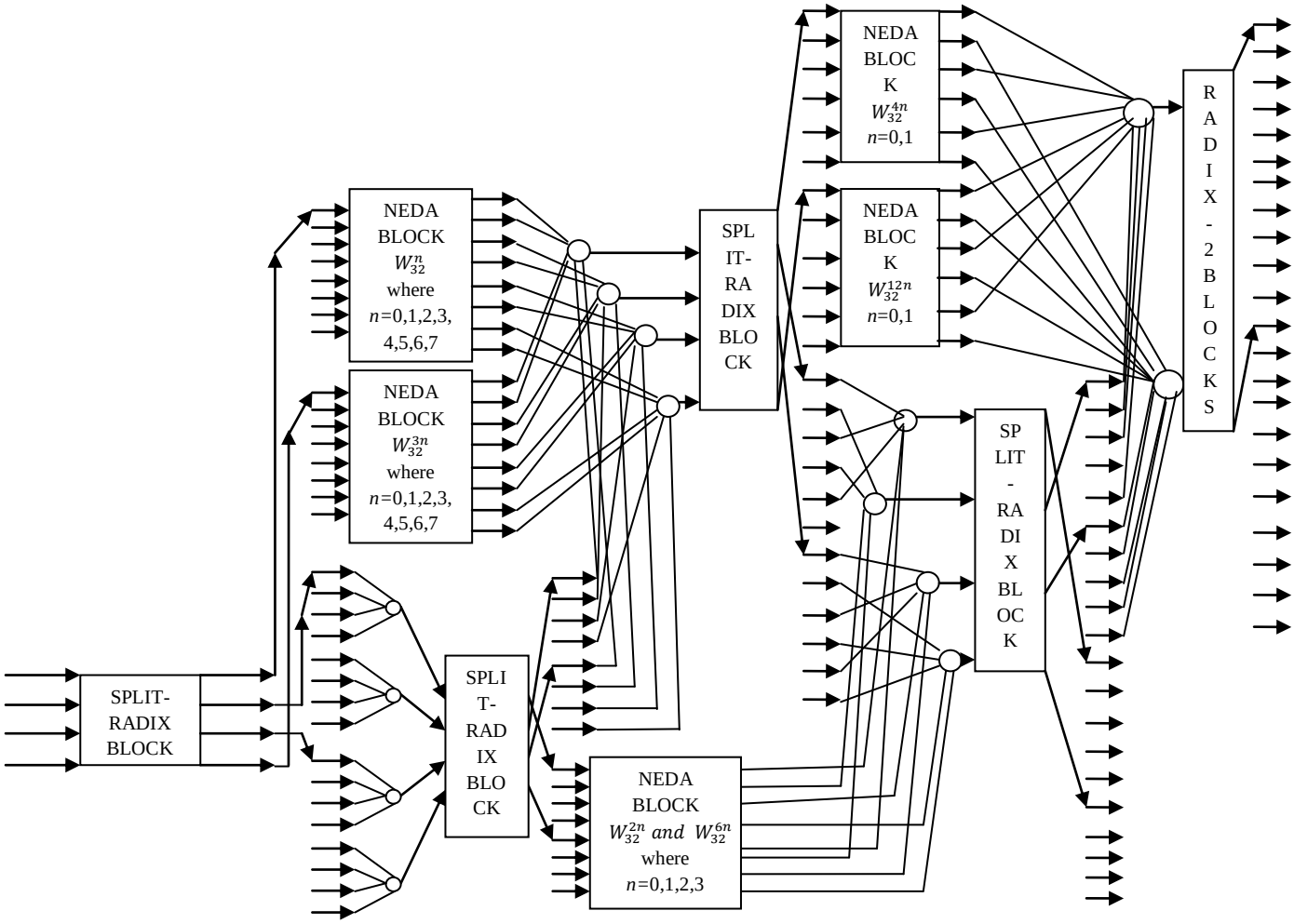

Figure 19. Proposed Architecture – II of 32-Point Split-Radix FFT

In table 7, the internal signals W0 to W15 are obtained after multiplying the signals P0 to P7 and P24 to P31 with their respective twiddle factors of second stage. Similarly, the signals T8, T9, T10, T11, T20, T21, T22 and T23 are obtained after multiplying the signals Q8, Q9, Q10, Q11, Q20, Q21, Q22 and Q23 with their corresponding twiddle factors of third stage. Finally, the signals L0, L1, L6, L7, L12, L13, L18, L19, L8, L9, L14 and L15 are obtained after multiplying the signals S0, S1, S6, S7, R12, R13, R18, R19, S8, S9, S14 and S15 with their twiddle factors of fourth stage respectively. The twiddle factor multiplications have been performed using NEDA blocks at respective stages. The outputs of the proposed architecture start coming from the 23rd clock cycle till 33rd clock cycle in bit-reversal order.

Table 7. Dataflow Table for Input-Outputs of Proposed Architecture – II of 32-point Split-Radix FFT

Clock Cycle	Inputs	Outputs
1	x0,x8,x16,x24	
2	x1,x9,x17,x25	P0,P8,P16,P24
3	x2,x10,x18,x26	P1,P9,P17,P25
4	x3,x11,x19,x27	P2,P10,P18,P26
5	x4,x12,x20,x28	P3,P11,P19,P27
6	x5,x13,x21,x29	P4,P12,P20,P28
7	x6,x14,x22,x30	P5,P13,P21,P29
8	x7,x15,x23,x31	P6,P14,P22,P30
9		P7,P15,P23,P31
10	P8,P12,P16,P20	
11	P9,P13,P17,P21	Q8,Q12,Q16,Q20
12	P10,P14,P18,P22	Q9,Q13,Q17,Q21
13	P11,P15,P19,P23	Q10,Q14,Q18,Q22
14		Q11,Q15,Q19,Q23
15	W0,W2,W4,W6	
16	W1,W3,W5,W7	S0,S2,S4,S6
17	Q12,Q14,Q16,Q18	S1,S3,S5,S7
18	Q13,Q15,Q17,Q19	R12,R14,R16,R18
19	W8,W10,W12,W14	R13,R15,R17,R19
20	W9,W11,W13,W15	S8,S10,S12,S14
21		S9,S11,S13,S15
22	S2,S3,S4,S5	
23	T8,T9,T10,T11	Y9,U3,U4,Y25
24	R14,R15,R16,R17	Y10,V9,V10,Y26
25	T20,T21,T22,T23	Y8,U15,U16,Y24
26	S10,S11,S12,S13	Y14,V21,V22,Y30
27		Y11,U11,U12,Y27
28	L0,L1,U3,U4	Y5,Y21,Y1,Y17
29	L6,L7,V9,V10	Y13,Y29,Y2,Y18
30	L12,L13,U15,U16	Y4,Y20,Y0,Y16
31	L18,L19,V21,V22	Y12,Y28,Y6,Y22
32	L8,L9,U11,U12	Y7,Y23,Y3,Y19
33	L14,L15,0,0	Y15,Y31,0,0

5.3.2 FPGA Device Utilization Summary

The proposed architectures have been implemented using Xilinx ISE as well as Altera Quartus II, wherever applicable. The proposed architecture – I can operate at a maximum frequency of 100.368 MHz on Xilinx Virtex-5 FPGAs. The outputs of proposed architecture – I are obtained after 45 ns, which results in its latency, in parallel. But, as the number of

IOBs is too high to accommodate, we go for proposed architecture – II. Table 8 shows the FPGA device utilization summary of proposed architecture – II. The power has been calculated using Xilinx XPower Analyzer.

Table 8. FPGA Device Utilization Summary of Proposed Architecture – II of 32-point Split-Radix FFT

FPGA device: XC5VLX330T-2FF1738	**Proposed Architecture – II**	
	Used	**Utilization**
Number of occupied slices	2426	51840 (4%)
Number of slice registers	5010	207360 (2%)
Number of slice LUTs	7099	207360 (3%)
Frequency	527.329 MHz	
Dynamic Power at maximum frequency	0.40262 W	

Many FFT designs have been reported using split-radix method and also utilizing the pipelining technique [37] – [38]. Table 9 shows the comparison results of the proposed architecture – II, with the architecture mentioned in [39]. The comparison has been made using Altera Quartus II and its Cyclone II family of FPGA. From table III, it is clear that, the proposed architecture gives better results in terms of speed, area, and power.

Table 9. Comparison of Proposed Architecture – II of 32-Point Split-Radix FFT Using Altera Cyclone II Family of FPGA

FPGA comparison results using Altera Cyclone II	**[39]**	**Proposed Architecture – II**
Number of inputs	32	32
Combinational functions	1442	14304
Logic registers	857	1123
18x18 multipliers	4	0
Memory	2(1K)	0
Execution time (µs)	7.995	0.14457
Frequency (MHz)	100	210.97
Device	EP2C35	EP2C70

5.3.3 ASIC Implementation Results

Table 10 shows the ASIC implementation of the proposed architectures in 0.18µm process technology using Synopsys DC for logic synthesis and Cadence SoC Encounter for physical

design. The process technology that has been followed to carryout physical design of the proposed architectures is UMC 0.18µm mixed mode generic core.

Table 10. ASIC Implementation Results of the Proposed Architectures of 32-Point Split-Radix FFT Using Synopsys DC and Cadence SoC Encounter

ASIC implementation results using Synopsys DC	**Process technology: 0.18µm**	
	Proposed Architecture – I	**Proposed Architecture – II**
Total cell area	1063769.421537	803245.469974
Total dynamic power	84.1841 mW	14.9286 mW
Add-sub width	16 bits	16 bits
Slack at 100 MHz	3.68 ns	6.62 ns

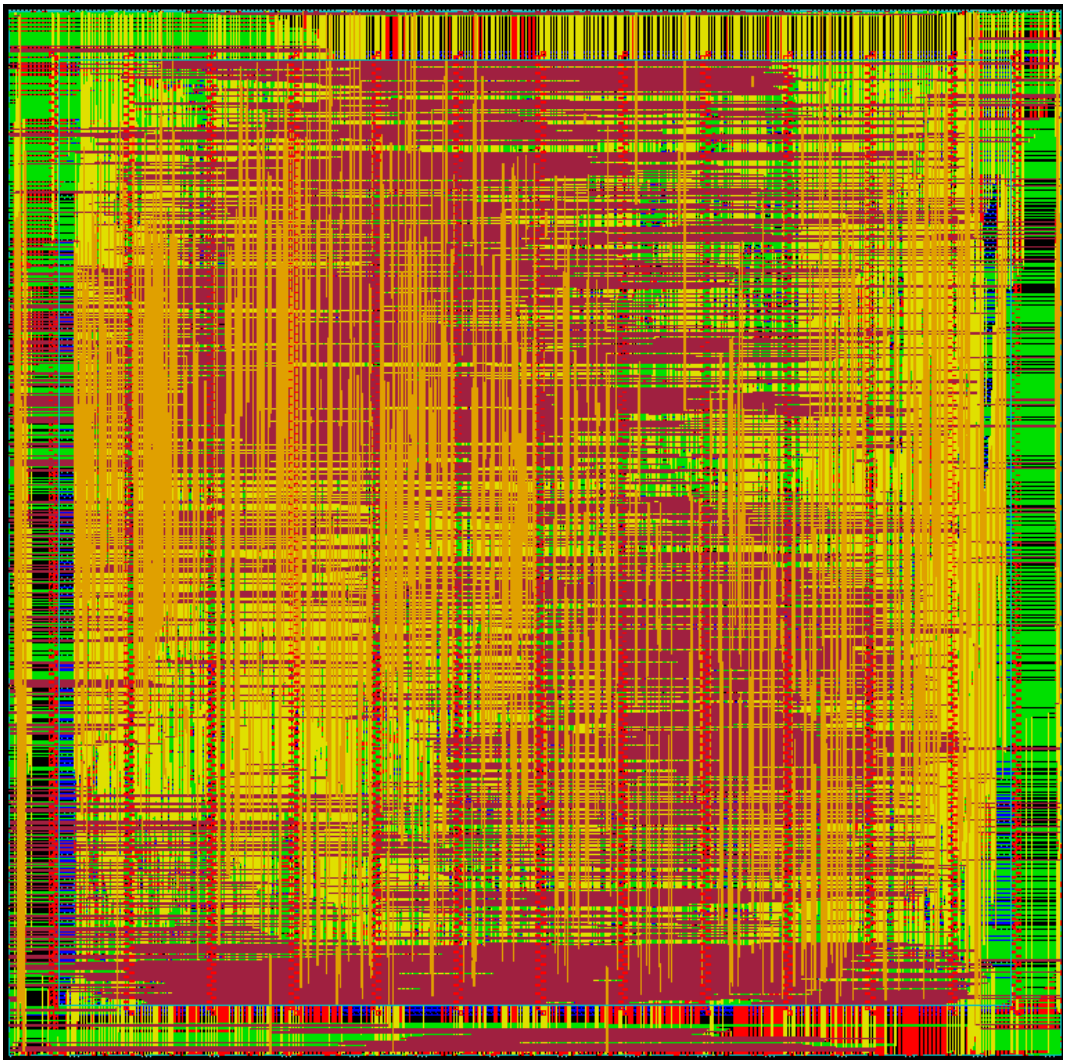

Figure 20. Physical Layout of Proposed Architecture – I of 32-Point Split-Radix FFT

The physical design of proposed architectures has been made in such a way that the timing constraints are met after both placement as well as routing. The layouts are shown in

figure 20 and figure 21. The core utilization of proposed designs has been set to 0.8 to avoid congestion while routing. The proposed architectures have been routed using Nano route. The slack achieved for proposed architecture – I at 100 MHz clock is 3.68 ns and for proposed architecture – II is 6.62 ns. From table IV it is clear proposed architecture – II gives better results in terms of area and power compared to proposed architecture – I.

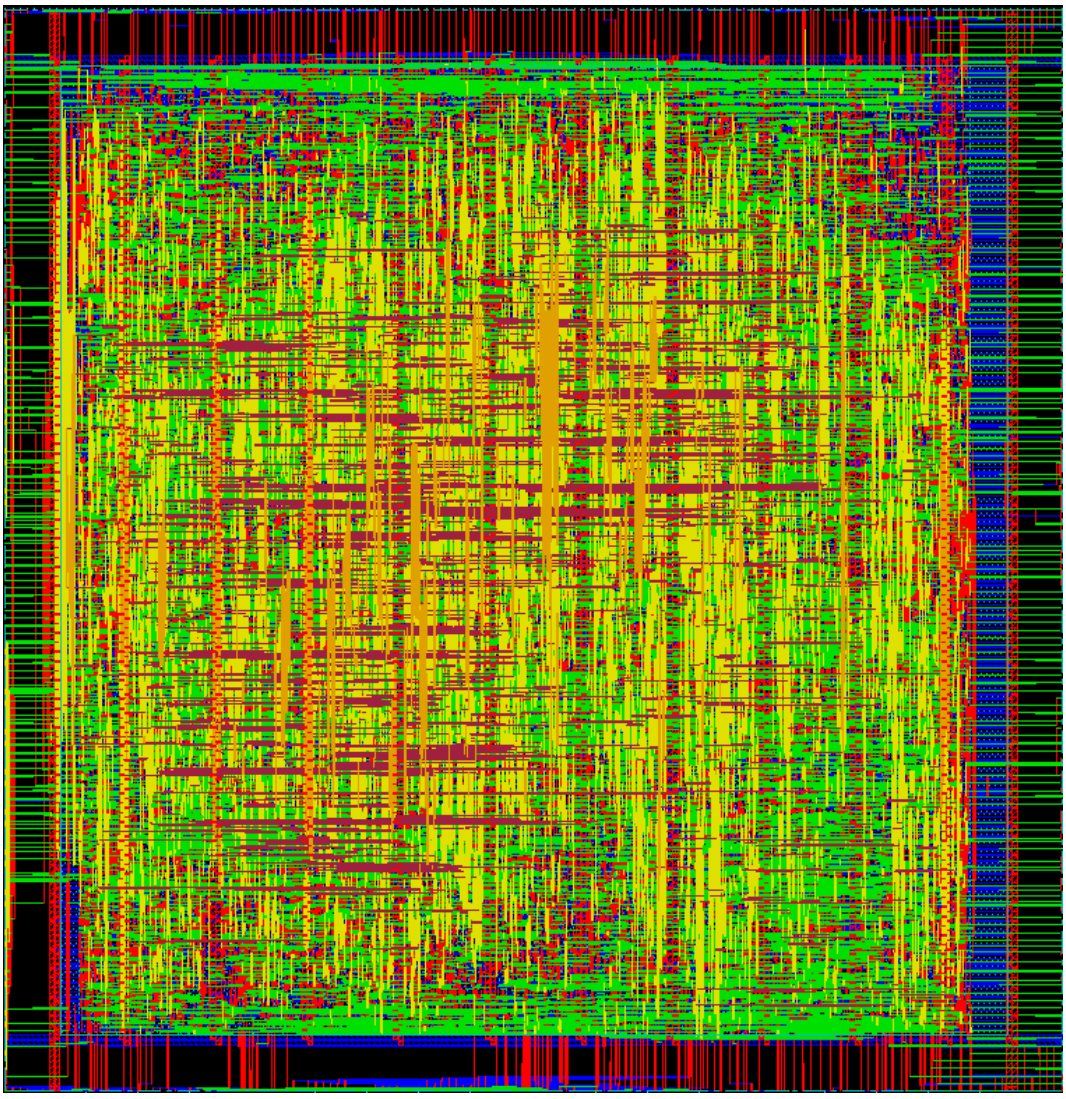

Figure 21. Physical Layout of Proposed Architecture – II of 32-Point Split-Radix FFT

5.4 64-Point Complex FFT Core

5.4.1 64-Point Complex FFT Core Using Radix-4 Algorithm

Radix-4 FFT algorithm is well known for its less complexity while retaining the simplicity of radix-2 FFT algorithm. The architecture for a 64-point complex FFT core using radix-4 algorithm has been proposed. NEDA technique has been used to implement twiddle multiplications. In order to reduce the number of IOBs and to increase speed and throughput, both folding and pipelining techniques have been used. Figure 22 shows the overview of the proposed architecture. It takes three stages to get the final output.

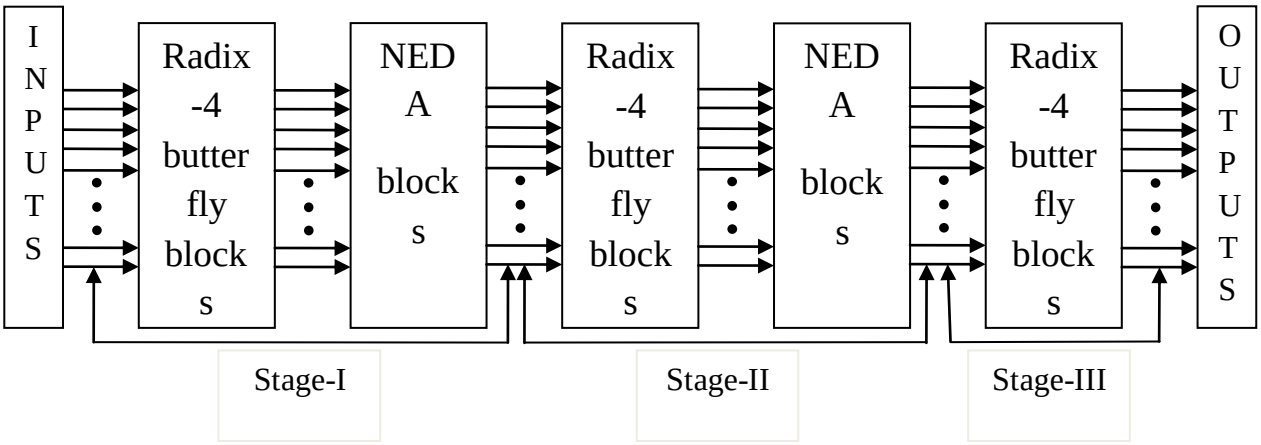

Figure 22. Overview of the Proposed Architecture for 64-Point FFT Core Using Radix-4 Algorithm

The proposed architecture has been folded by a factor of 4. The length of the data bus for inputs and outputs have confined to 16. Table 11 shows the dataflow for input-output of the proposed architecture.

Table 11. Dataflow Table for Input-Outputs of Proposed Architecture of 64-Point FFT Core Using Radix-4 Algorithm

Clock cycle	Inputs	Outputs
1	x0,x16,x32,x48	
2	x1,x17,x33,x49	S1_0,S1_16,S1_32,S1_48
3	x2,x18,x34,x50	S1_1,S1_17,S1_33,S1_49
4	x3,x19,x35,x51	S1_2,S1_18,S1_34,S1_50

5	x4,x20,x36,x52	S1_3,S1_19,S1_35,S1_51
6	x5,x21,x37,x53	S1_4,S1_20,S1_36,S1_52
7	x6,x22,x38,x54	S1_5,S1_21,S1_37,S1_53
8	x7,x23,x39,x55	S1_6,S1_22,S1_38,S1_54
9	x8,x24,x40,x56	S1_7,S1_23,S1_39,S1_55
10	x9,x25,x41,x57	S1_8,S1_24,S1_40,S1_56
11	x10,x26,x42,x58	S1_9,S1_25,S1_41,S1_57
12	x11,x27,x43,x59	S1_10,S1_26,S1_42,S1_58
13	x12,x28,x44,x60	S1_11,S1_27,S1_43,S1_59
14	x13,x29,x45,x61	S1_12,S1_28,S1_44,S1_60
15	x14,x30,x46,x62	S1_13,S1_29,S1_45,S1_61
16	x15,x31,x47,x63	S1_14,S1_30,S1_46,S1_62
17		S1_15,S1_31,S1_47,S1_63
18	S1_0, S1_4, S1_8, S1_12	
19	S1_1, S1_5, S1_9, S1_13	S2_0, S2_4, S2_8, S2_12
20	S1_2, S1_6, S1_10, S1_14	S2_1, S2_5, S2_9, S2_13
21	S1_3, S1_7, S1_11, S1_15	S2_2, S2_6, S2_10, S2_14
22	S1_16,S1_N_20,S1_N_24,S1_N_28	S2_3, S2_7, S2_11, S2_15
23	S1_N_17,S1_N_21,S1_N_25,S1_N_29	S2_16,S2_20,S2_24,S2_28
24	S1_N_18,S1_N_22,S1_N_26,S1_N_30	S2_17,S2_21,S2_25,S2_29
25	S1_N_19,S1_N_23,S1_N_27,S1_N_31	S2_18,S2_22,S2_26,S2_30
26	S1_32,S1_N_36,S1_N_40,S1_N_44	S2_19,S2_23,S2_27,S2_31
27	S1_N_33,S1_N_37,S1_N_41,S1_N_45	S2_32,S2_36,S2_40,S2_44
28	S1_N_34,S1_N_38,S1_N_42,S1_N_46	S2_33,S2_37,S2_41,S2_45
29	S1_N_35,S1_N_39,S1_N_43,S1_N_47	S2_34,S2_38,S2_42,S2_46
30	S1_48,S1_N_52,S1_N_56,S1_N_60	S2_35,S2_39,S2_43,S2_47
31	S1_N_49,S1_N_53,S1_N_57,S1_N_61	S2_48,S2_52,S2_56,S2_60
32	S1_N_50,S1_N_54,S1_N_58,S1_N_62	S2_49,S2_53,S2_57,S2_61
33	S1_N_51,S1_N_55,S1_N_59,S1_N_63	S2_50,S2_54,S2_58,S2_62
34		S2_51,S2_55,S2_59,S2_63
35	S2_0, S2_1, S2_2, S2_3	
36	S2_4, S2_N_5, S2_N_6, S2_N_7	OUT0,OUT16,OUT32,OUT48

37	S2_8, S2_N_9, S2_N_10, S2_N_11	OUT1,OUT17,OUT33,OUT49
38	S2_12, S2_N_13, S2_N_14, S2_N_15	OUT2,OUT18,OUT34,OUT50
39	S2_16, S2_17, S2_18, S2_19	OUT3,OUT19,OUT35,OUT51
40	S2_20, S2_N_21, S2_N_22, S2_N_23	OUT4,OUT20,OUT36,OUT52
41	S2_24, S2_N_25, S2_N_26, S2_N_27	OUT5,OUT21,OUT37,OUT53
42	S2_28, S2_N_29, S2_N_30, S2_N_31	OUT6,OUT22,OUT38,OUT54
43	S2_32, S2_33, S2_34, S2_35	OUT7,OUT23,OUT39,OUT55
44	S2_36, S2_N_37, S2_N_38, S2_N_39	OUT8,OUT24,OUT40,OUT56
45	S2_40, S2_N_41, S2_N_42, S2_N_43	OUT9,OUT25,OUT41,OUT57
46	S2_44, S2_N_45, S2_N_46, S2_N_47	OUT10,OUT26,OUT42,OUT58
47	S2_48, S2_49, S2_50, S2_51	OUT11,OUT27,OUT43,OUT59
48	S2_52, S2_N_53, S2_N_54, S2_N_55	OUT12,OUT28,OUT44,OUT60
49	S2_56, S2_N_57, S2_N_58, S2_N_59	OUT13,OUT29,OUT45,OUT61
50	S2_60, S2_N_61, S2_N_62, S2_N_63	OUT14,OUT30,OUT46,OUT62
51		OUT15,OUT31,OUT47,OUT63

In the first 16 clock cycles, all the 64 inputs are fed to the first stage radix-4 butterfly block. The outputs of the first stage butterfly blocks are stored in registers and then multiplied by their respective twiddle factors as shown in the figure 23. In table 11, the above steps give transition of S1_datasample to S1_N_datasample. After performing the twiddle multiplications by NEDA technique, the outputs of the stage I are fed to radix-4 butterfly blocks present in stage II in an order as given in table 11. Again the outputs of stage II butterfly blocks are stored in registers and later, those are multiplied by their respective twiddle factors as shown in figure 23.

In table 11, the above steps give transition of S1_N_datasample to S2_datasample to S2_N_datasample. After performing the twiddle multiplications by NEDA technique, the outputs of the stage I are fed to radix-4 butterfly blocks present in stage III in an order as given in table 11. In this stage, NEDA blocks are not required as the samples are multiplied by W_{64}^{0} (that is '1'). We get the final outputs after the third stage in an order similar to the order of the inputs. Table 12 gives complete information about the inputs to the radix-4 butterfly blocks and the twiddle multiplications in each stage.

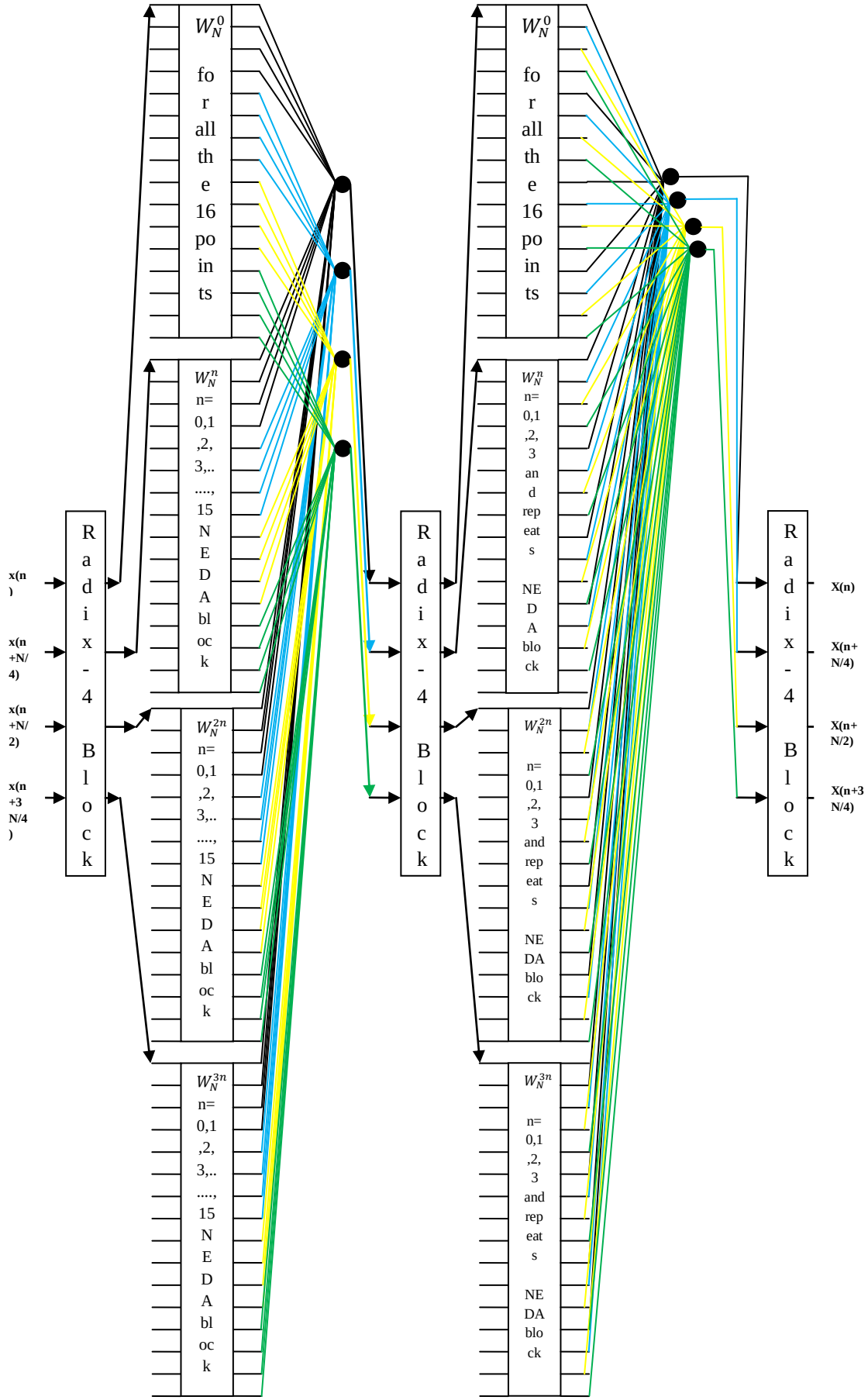
x(n)
x(n+N/4)
x(n+N/2)
x(n+3N/4)
Radix-4 Block
W_N^0 for all the 16 points
W_N^n n= 0,1,2,3,....,15 NEDA block
W_N^{2n} n= 0,1,2,3,....,15 NEDA block
W_N^{3n} n= 0,1,2,3,....,15 NEDA block
Radix-4 Block
W_N^0 for all the 16 points
W_N^n n= 0,1,2,3 and repeats NEDA block
W_N^{2n} n= 0,1,2,3 and repeats NEDA block
W_N^{3n} n= 0,1,2,3 and repeats NEDA block
Radix-4 Block
X(n)
X(n+N/4)
X(n+N/2)
X(n+3N/4)

Figure 23. Block Diagram of the Proposed Architecture for 64-Point FFT Core Using Radix-4 Algorithm

Table 12. Inputs to the Radix-4 Blocks and Twiddle Factors for Data Samples in Each Stage of 64-Point FFT Core Using Radix-4

Stage Number	**Inputs to the Radix-4 Blocks**	**Twiddle Factors to be Multiplied**
Stage-I	x(n), x(n+N/4), x(n+N/2), x(n+3N/4) where n=0, 1, 2, ..., (N/4-1) and N=64	Data samples 0 to 15: W_N^0 Data samples 16 to 31: W_N^n Data samples 32 to 47: W_N^{2n} Data samples 48 to 63: W_N^{3n} where n=0, 1, 2, ..., (N/4-1) and N=64
Stage-II	x(n+M), x(n+N/4+M), x(n+N/2+M), x(n+3N/4+M) where n=0, 1, 2,3 and N=16 M=0, 16, 32, 48 and the value change only after n changes from 0 to 3 for each value of M	Data samples 0 to 3, 16 to 19, 32 to 35, 48 to 51: W_N^0 Data samples 4 to 7, 20 to 23, 36 to 39, 52 to 55: W_N^n Data samples 8 to 11, 24 to 27, 40 to 43, 56 to 59: W_N^{2n} Data samples 12 to 15, 28 to 31, 44 to 47, 60 to 63: W_N^{3n} where n=0, 1, 2, 3 and N=64
Stage-III	x(4n), x(4n+1), x(4n+2), x(4n+3) where n=0, 1, 2, ..., (N/4-1) and N=64	All the data samples are multiplied with W_N^0 where N=64

5.4.2 64-Point Complex FFT Core Using Radix-8 Algorithm

The proposed architecture computes 64-points complex Fast Fourier Transform (FFT) using radix-8 algorithm. Higher radix FFT algorithms have the traditional advantage of using less number of computational elements. It takes two stages to compute the FFT of 64-points using radix-8 algorithm. As shown in the figure 24, in stage-I, the inputs are given to radix-8

butterfly blocks and the output of radix-8 butterfly blocks are multiplied with their respective twiddle factors. The twiddle multiplication has been performed by using NEDA technique in the proposed design. The output of NEDA blocks are given to the radix-8 butterfly blocks present in stage-II. The final outputs are taken after stage-II. The inputs and outputs are taken in same order.

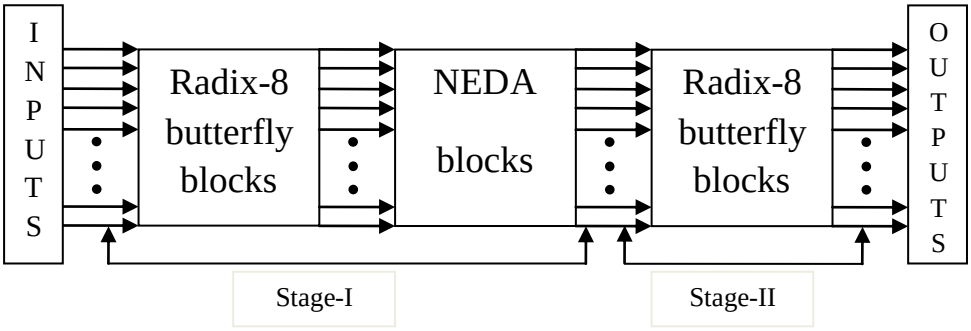

Figure 24. Overview of the Proposed Architecture for 64-Point FFT Core Using Radix-8 Algorithm

The precession of all inputs and outputs is of 16 bits. The proposed architecture has also used folding transformation by a factor of eight. Pipelining method has also been used in the design as shown in figure 25. In stage-I, at each clock cycle, eight inputs are given to radix-8 butterfly block and the outputs are stored in a predefined matrix. This process continues for eight clock cycles, so that all 64 inputs are being covered. The outputs are given to NEDA blocks for twiddle multiplication. The output of NEDA blocks are given to stage-II radix-8 butterfly block. Eight outputs of NEDA block are given to a radix-8 butterfly block in a clock cycle. This continues for eight clock cycles. The total outputs are calculated after eight clock cycles in stage-II. The proposed architecture, shown in figure 25, takes 8 inputs at a time which sums up to 8 clock cycles to read all the 64 inputs. For every clock cycle, the outputs of the first stage radix-8 block are stored in registers and this process continues till all 64 outputs are stored. Later, the stored outputs are processed for second stage computations which consist of either NEDA blocks. The outputs of the NEDA blocks are fed to second stage radix-8 blocks.

All the 64 inputs are fed to the first stage radix-8 block in first 8 clock cycles as shown in table 13. The outputs of the first stage radix-8 blocks are stored in registers for further processing. In the next 7 clock cycles, the outputs of the first stage radix-8 block are multiplied by their respective twiddle factors as shown in the figure 25. The final output starts

coming from 17th clock cycle till 24th clock cycle in the order the input has been fed. We have further minimized the number of clock cycles for getting the final output by assigning less number of clock cycles for the NEDA block.

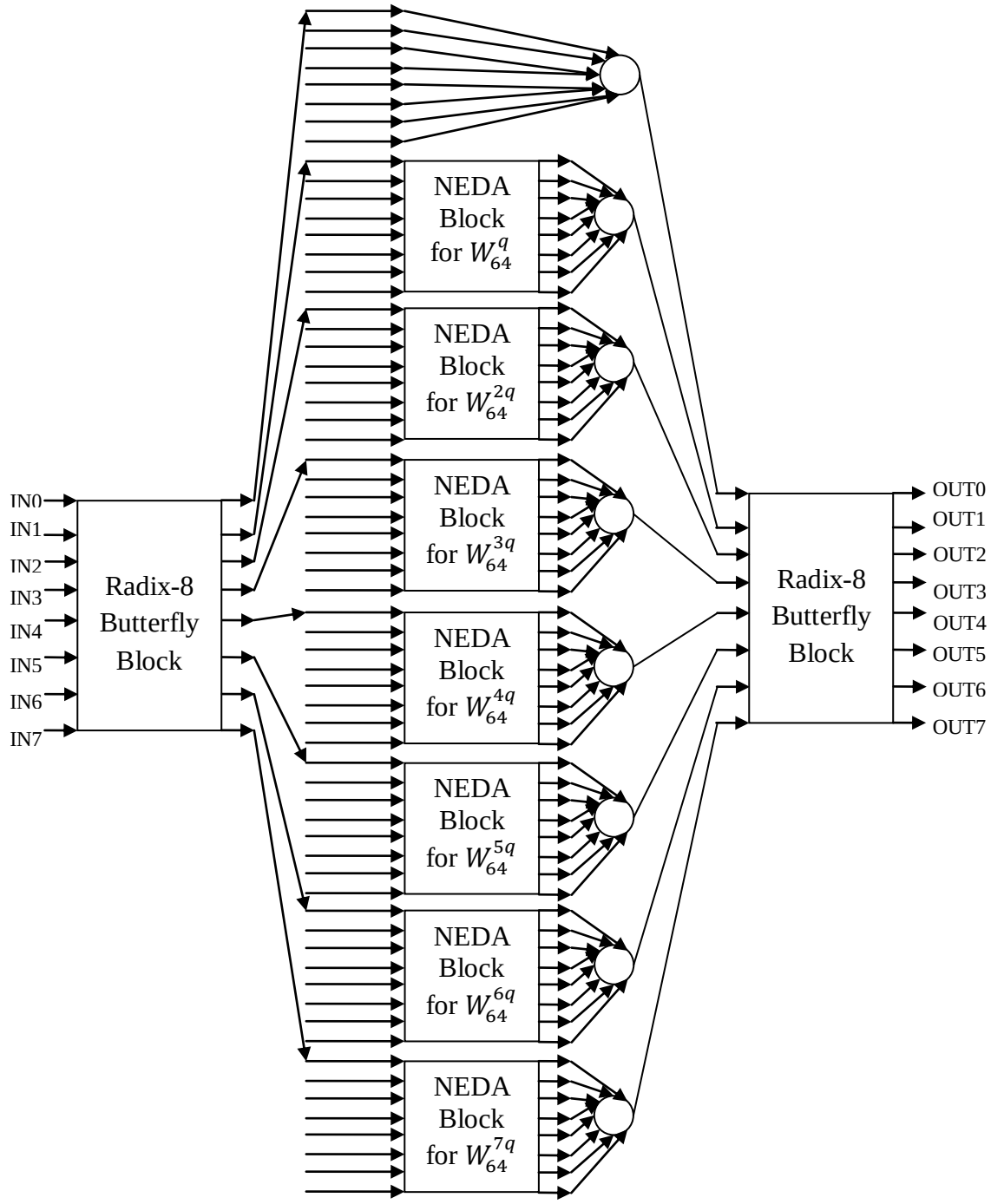

Figure 25. Block Diagram of the Proposed Architecture for 64-Point FFT Core Using Radix-8 Algorithm (q=0, 1, 2, 3, 4, 5, 6, and 7 for each case)

Table 13. Dataflow Table for Input-Outputs of Proposed Architecture of 64-Point FFT Core Using Radix-8 Algorithm

Clock cycle	**Inputs**	**Outputs**
1	x0,x8,x16,x24,x32,x40,x48,x56	
2	x1,x9,x17,x25,x33,x41,x49,x57	S1_0,S1_8,S1_16,S1_24,S1_32,S1_40,S1_48,S1_56
3	x2,x10,x18,x26,x34,x42,x50,x58	S1_1,S1_9,S1_17,S1_25,S1_33,S1_41,S1_49,S1_57
4	x3,x11,x19,x27,x35,x43,x51,x59	S1_2,S1_10,S1_18,S1_26,S1_34,S1_42,S1_50,S1_58
5	x4,x12,x20,x28,x36,x44,x52,x60	S1_3,S1_11,S1_19,S1_27,S1_35,S1_43,S1_51,S1_59
6	x5,x13,x21,x29,x37,x45,x53,x61	S1_4,S1_12,S1_20,S1_28,S1_36,S1_44,S1_52,S1_60
7	x6,x14,x22,x30,x38,x46,x54,x62	S1_5,S1_13,S1_21,S1_29,S1_37,S1_45,S1_53,S1_61
8	x7,x15,x23,x31,x39,x47,x55,x63	S1_6,S1_14,S1_22,S1_30,S1_38,S1_46,S1_54,S1_62
9	S1_1,S1_9,S1_17,S1_25,S1_33,S1_41,S1_49,S1_57	S1_7,S1_15,S1_23,S1_31,S1_39,S1_47,S1_55,S1_63
10	S1_2,S1_10,S1_18,S1_26,S1_34,S1_42,S1_50,S1_58	S1_N_1,S1_N_9,S1_N_17,S1_N_25,S1_N_33,S1_N_41,S1_N_49,S1_N_57
11	S1_3,S1_11,S1_19,S1_27,S1_35,S1_43,S1_51,S1_59	S1_N_2,S1_N_10,S1_N_18,S1_N_26,S1_N_34,S1_N_42,S1_N_50,S1_N_58
12	S1_4,S1_12,S1_20,S1_28,S1_36,S1_44,S1_52,S1_60	S1_N_3,S1_N_11,S1_N_19,S1_N_27,S1_N_35,S1_N_43,S1_N_51,S1_N_59
13	S1_5,S1_13,S1_21,S1_29,S1_37,S1_45,S1_53,S1_61	S1_N_4,S1_N_12,S1_N_20,S1_N_28,S1_N_36,S1_N_44,S1_N_52,S1_N_60

14	S1_6,S1_14,S1_22,S1_30,S1_38,S1_46,S1_54,S1_62	S1_N_5,S1_N_13,S1_N_21,S1_N_29,S1_N_37,S1_N_45,S1_N_53,S1_N_61
15	S1_7,S1_15,S1_23,S1_31,S1_39,S1_47,S1_55,S1_63	S1_N_6,S1_N_14,S1_N_22,S1_N_30,S1_N_38,S1_N_46,S1_N_54,S1_N_62
16	S1_0,S1_8,S1_16,S1_24,S1_32,S1_40,S1_48,S1_56	S1_N_7,S1_N_15,S1_N_23,S1_N_31,S1_N_39,S1_N_47,S1_N_55,S1_N_63
17	S1_N_1,S1_N_9,S1_N_17,S1_N_25,S1_N_33,S1_N_41,S1_N_49,S1_N_57	OUT0,OUT8,OUT16,OUT24,OUT32,OUT40,OUT48,OUT56
18	S1_N_2,S1_N_10,S1_N_18,S1_N_26,S1_N_34,S1_N_42,S1_N_50,S1_N_58	OUT1,OUT9,OUT17,OUT25,OUT33,OUT41,OUT49,OUT57
19	S1_N_3,S1_N_11,S1_N_19,S1_N_27,S1_N_35,S1_N_43,S1_N_51,S1_N_59	OUT2,OUT10,OUT18,OUT26,OUT34,OUT42,OUT50,OUT58
20	S1_N_4,S1_N_12,S1_N_20,S1_N_28,S1_N_36,S1_N_44,S1_N_52,S1_N_60	OUT3,OUT11,OUT19,OUT27,OUT35,OUT43,OUT51,OUT59
21	S1_N_5,S1_N_13,S1_N_21,S1_N_29,S1_N_37,S1_N_45,S1_N_53,S1_N_61	OUT4,OUT12,OUT20,OUT28,OUT36,OUT44,OUT52,OUT60
22	S1_N_6,S1_N_14,S1_N_22,S1_N_30,S1_N_38,S1_N_46,S1_N_54,S1_N_62	OUT5,OUT13,OUT21,OUT29,OUT37,OUT45,OUT53,OUT61
23	S1_N_7,S1_N_15,S1_N_23,S1_N_31,S1_N_39,S1_N_47,S1_N_55,S1_N_63	OUT6,OUT14,OUT22,OUT30,OUT38,OUT46,OUT54,OUT62
24		OUT7,OUT15,OUT23,OUT31,OUT39,OUT47,OUT55,OUT63

5.4.3 Comparison between the Proposed Architectures

5.4.3.1 FPGA Device Utilization Summary

The proposed architectures have been implemented using Xilinx ISE. The proposed architecture can operate at a maximum frequency of 327.284 MHz and 447.838 MHz on Xilinx Virtex-5 FPGAs for the proposed FFT core using radix-4 and radix-8 respectively. The outputs of proposed architecture are obtained after 36 and 15 clock cycles for the proposed FFT core using radix-4 and radix-8 respectively. Table 14 shows the FPGA device utilization summary of proposed architecture. The power has been calculated using Xilinx

XPower Analyzer. It can be concluded from table 14 that the proposed FFT core using radix-8 algorithm gives better performance in terms of speed and throughput but consumes more power than the proposed FFT core using radix-4 algorithm.

Table 14. FPGA Device Utilization Summary of Proposed Architectures of 64-Point FFT Core

FPGA device: XC5VLX330T-2FF1738	**Using Radix-4 FFT Algorithm**	**Using Radix-8 FFT Algorithm**
Number of adders	1784	1488
Number of registers	823	279
Data path size	16 bits	16 bits
Maximum frequency	327.284 MHz	447.838 MHz
Throughput	5236.544 Ms/S	7165.408 Ms/S
Dynamic Power at maximum frequency	0.35892 W	1.15023 W

5.4.3.2 ASIC Implementation Results

Table 15 shows the ASIC implementation of the proposed architectures in 0.18µm process technology using Synopsys DC for logic synthesis and Cadence SoC Encounter for physical design. The process technology that has been followed to carryout physical design of the proposed architectures is UMC 0.18µm mixed mode generic core. The physical design of proposed architectures has been made in such a way that the timing constraints are met after both placement as well as routing. The physical layouts are shown in figure 26 and 27 for the proposed FFT core using radix-4 and radix-8 respectively. The core utilization of proposed designs has been set to 0.8 to avoid congestion while routing. The proposed architectures have been routed using Nano route. The slack achieved for proposed architectures at 100 MHz clock are 6.29 ns and 7.24 ns for the proposed FFT core using radix-4 and radix-8 respectively. It can be concluded from the table 15 that the FFT core using radix-8 algorithm consumes more power but it is efficient in terms of speed and area than the proposed FFT core using radix-4 algorithm.

Table 15. ASIC Implementation Results of the Proposed Architectures of 64-Point FFT Core Using Synopsys DC and Cadence SoC Encounter

ASIC implementation results using Synopsys DC	**Using Radix-4 FFT Algorithm**	**Using Radix-8 FFT Algorithm**
Total cell area	1832064.003531	1479439.695747
Total dynamic power	21.0620 mW	39.3705 mW
Add-sub width	16 bits	16 bits
Slack at 100 MHz	6.29 ns	7.24 ns

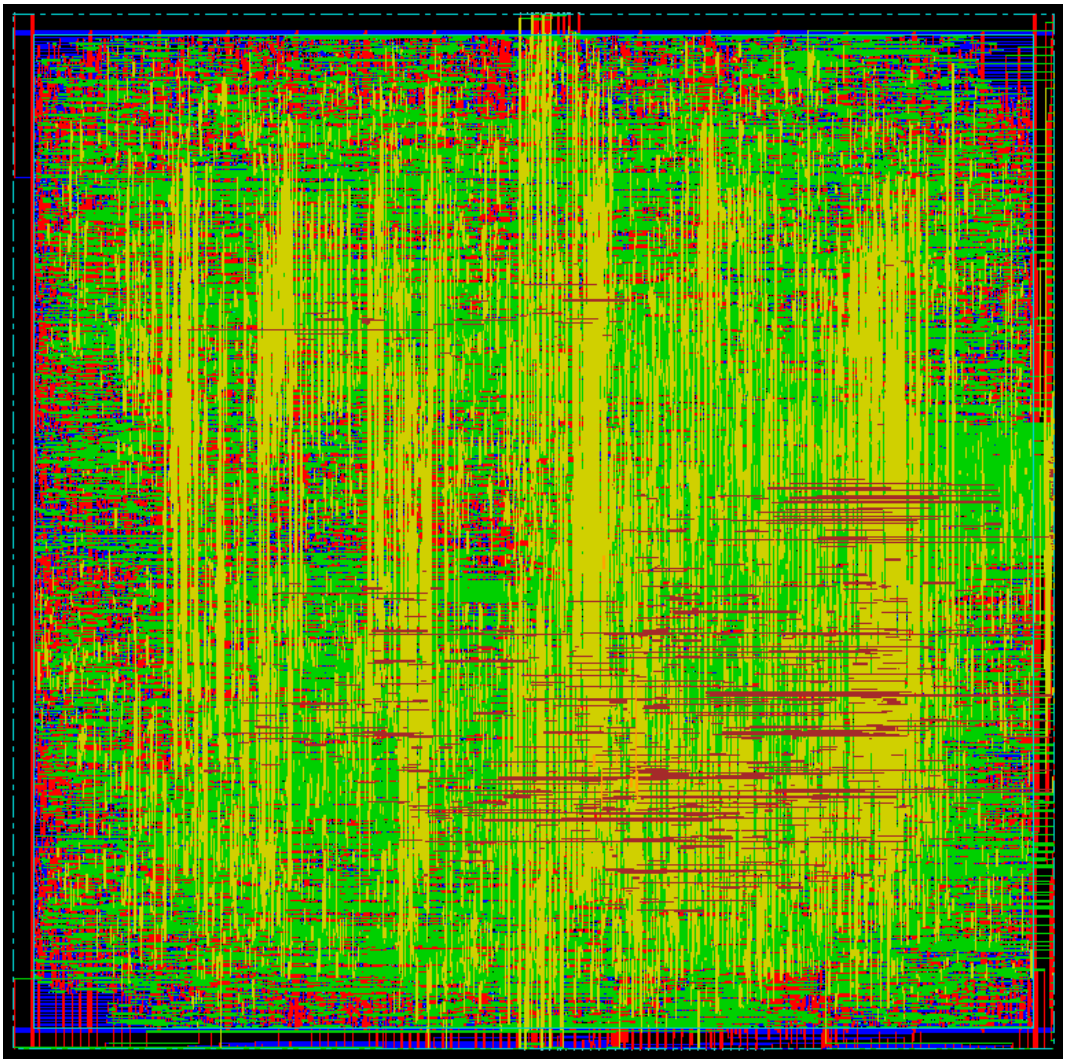

Figure 26. Physical Layout of the Proposed Architecture for 64-Point FFT Core Using Radix-4 Algorithm

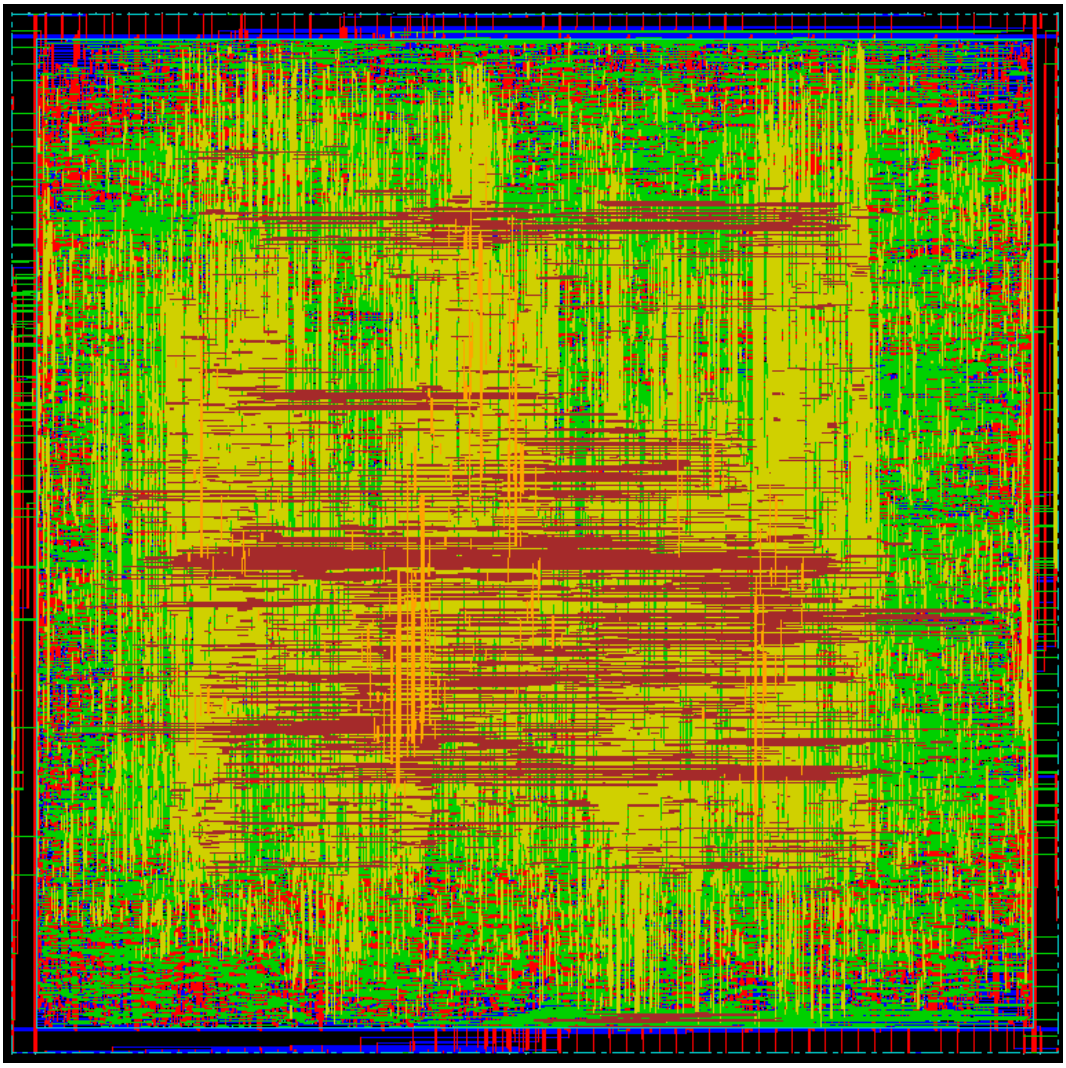

Figure 27. Physical Layout of the Proposed Architecture for 64-Point FFT Core Using Radix-8 Algorithm

5.4.4 Comparison with Existing Architectures

The above proposed architectures for 64-point complex FFT core has been compared with different commercially available IP cores and existing designs in terms of power consumption, area, and number of clock cycles to get the final output. Table 16 gives a comparative study of number of clock cycles to get the final output between the proposed architectures and the existing designs and IP cores. It can be concluded that the proposed FFT core using radix-8 algorithm gives output earlier than the others. From the table 17 it is clear that the proposed designs gives better results in terms of power and area with respect to the other existing designs. In the table 16 and 17, proposed design 1 and 2 refers to the proposed FFT core using radix-4 and radix-8 algorithms respectively.

Table 16. Performance Comparison of the Proposed FFT Processors with the Commercially Available 64-Point FFT/IFFT IP Cores and Existing Designs

Designs	Word Length	Number of Clock Cycles
IP Core[40]	16	192
IP Core [41]	12	112
IP Core [42]	16	1536
IP Core [43]	16	96
[44]	16	23
[45]	8	24
[46]	-	178
Proposed Design 1	16	36
Proposed Design 2	16	17

Table 17. Performance Comparison of the Proposed Designs with Existing Chipsets for Computing 64-Point FFT/IFFT

Processor	Word Length	Cycles Required	Technology (µm)	Power (W)	Area (mm^2)
[44]	16	23	0.25	0.041	6.8 (Core)
[47]	24	130	0.35	1.3	Core
[48]	16	208	2	1	282
[49]	16	222	0.75	-	156
[50]	16	-	0.6	-	62.4
[51]	32	-	0.18	0.069	-
[52]	-	-	0.18	0.217	-
Proposed Design 1	16	36	0.18	0.021	1.832
Proposed Design 2	16	17	0.18	0.0393	1.479

5.5 512-Point Complex FFT Core Using Radix-8 Algorithm

Finally, the architecture of 512-point complex FFT core [53] – [55] has been proposed using radix-8 algorithm. Both folding and pipelining techniques have been used in the design. NEDA technique has been used for twiddle multiplications in order to make the design multiplier-less. Figure 28 gives the overview of the architecture. It takes three stages to compute the final output. The inputs and outputs are in same order. The idea behind the proposed architecture is same as the previous architectures in the way that feed the input to the radix-8 blocks, store the outputs of the radix-8 blocks in registers and then multiply twiddle factors with the stored values using NEDA technique.

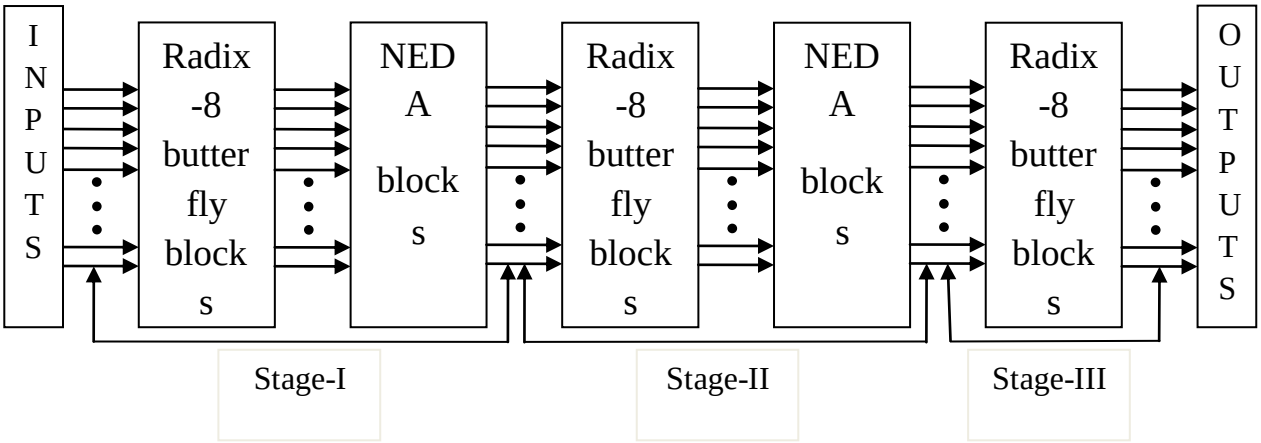

Figure 28. Overview of the Proposed Architecture for 512-Point FFT Core Using Radix-8 Algorithm

The proposed architecture has been folded by a factor of eight. Eight inputs are given to the first stage radix-8 blocks in one clock cycles, making it 64 clock cycles to feed all the 512 inputs to the first stage radix-8 blocks. The outputs of the first stage radix-8 blocks are stored in registers and are multiplied with their respective twiddle values using NEDA blocks. The outputs of first stage NEDA blocks are fed to second stage radix-8 blocks, eight in one clock cycle. The outputs of the second stage radix-8 blocks are stored in registers and are multiplied with their respective twiddle factors using NEDA technique. The outputs of the second stage NEDA blocks are fed to third stage radix-8 blocks, eight in one clock cycle. We get the final output after the third stage radix-8 blocks. Table 18 provides the information about the inputs to the radix-8 blocks and the twiddle factors to be multiplied with data samples in each stage.

Table 18. Inputs to the Radix-8 Blocks and Twiddle Factors for Data Samples in Each Stage of 512-Point Complex FFT Core Using Radix-8

Stage Number	**Inputs to the Radix-8 Blocks**	**Twiddle Factors to be Multiplied**
Stage-I	x(n), x(n+N/8), x(n+N/4), x(n+3N/8), x(n+N/2), x(n+5N/8), x(n+3N/4), x(n+7N/8) where n=0, 1, 2, ..., (N/8-1) and N=512	Data samples 0 to 63: W_N^0 Data samples 64 to 127: W_N^n Data samples 128 to 191: W_N^{2n} Data samples 192 to 255: W_N^{3n} Data samples 256 to 319: W_N^{4n} Data samples 320 to 383: W_N^{5n} Data samples 384 to 447: W_N^{6n} Data samples 448 to 511: W_N^{7n} where n=0, 1, 2, ..., (N/8-1) and N=512
Stage-II	x(n+M), x(n+N/8+M), x(n+N/4+M), x(n+3N/8+M), x(n+N/2+M), x(n+5N/8+M), x(n+3N/4+M), x(n+7N/8+M) where n=0, 1, 2, ..., (N/8-1) and N=64 M=0, 64, 128, 192, 256, 320, 384, 448 and the value change only after n changes from 0 to 7 for each value of M	Data samples 0 to 7, 64 to 71, 128 to 135, 192 to 199, 256 to 263, 320 to 327, 384 to 391, 448 to 455: W_N^0 Data samples 8 to 15, 72 to 79, 136 to 143, 200 to 207, 264 to 271, 328 to 335, 392 to 399, 456 to 463: W_N^n Data samples 16 to 23, 80 to 87, 144 to 151, 208 to 215, 272 to 279, 336 to 343, 400 to 407, 464 to 471: W_N^{2n} Data samples 24 to 31, 88 to 95, 152 to 159, 216 to 223, 280 to 287, 344 to 351, 408 to 415, 472 to 479: W_N^{3n} Data samples 32 to 39, 96 to 103, 160 to 167, 224 to 231, 288 to 295, 352 to 359, 416 to 423, 480 to 487: W_N^{4n} Data samples 40 to 47, 104 to 111, 168 to 175, 232 to 239, 296 to 303, 360 to 367, 424 to 431, 488 to 495:

		W_N^{5n} Data samples 48 to 55, 112 to 119, 176 to 183, 240 to 247, 304 to 311, 368 to 375, 432 to 439, 496 to 503: W_N^{6n} Data samples 56 to 63, 120 to 127, 184 to 191, 248 to 255, 312 to 319, 376 to 383, 440 to 447, 504 to 511: W_N^{7n} where n=0, 1, 2, 3, 4, 5, 6, 7 and N=512
Stage-III	x(8n), x(8n+1), x(8n+2), x(8n+3), x(8n+4), x(8n+5), x(8n+6), x(8n+7) where n=0, 1, 2, ..., (N/8-1) and N=512	All the data samples are multiplied with W_N^0 where N=512

Chapter 6

Conclusions

The Key Contributions

Future Research

Chapter 6

Conclusions

While designing various FFT cores, due to the use of multiplexers, memory, or ROMs, there is a substantial increase in power consumption and area. In order to increase speed and throughput, folding and pipelining methods have been approached by various existing designs. But the prime disadvantage of those architectures is the use of multipliers for twiddle multiplications. This present work has proposed various multiplier-less FFT cores using NEDA technique. Both folding and pipelining techniques have also been used in the proposed designs.

By doing such, it can be concluded that there is a substantial increase in the performance of the proposed FFT cores in terms of speed, power and area as compared to other existing architectures. Radix-4, split-radix and radix-8 FFT algorithms have been used in the different proposed architectures. The proposed architectures have been implemented in FPGA and ASIC. ASIC implementation of proposed architectures has been done using Synopsys and Cadence tools. Three different FFT cores have been proposed and have undergone complete FPGA and ASIC flow. Architecture for real-time implementation of FFT using LabVIEW and CompactRIO has also been proposed. Finally the architecture for 512-point FFT core using radix-8 has been proposed.

6.1 The Key Contributions

Radix-4, split-radix and radix-8 butterfly blocks have been optimized for better performance. NEDA technique has been used in the radix-8 butterfly block for twiddle multiplication. The device utilization summary on Xilinx ISE and the ASIC implementation results are also given in table 1 and 2 respectively. The physical layouts of the different butter blocks are also shown through figure 5 – 7 respectively.

The real-time implementation of 256-point FFT and finding the power spectrum using LabVIEW and CompactRIO has also been proposed. The final synthesis report has been given in table 6. Algorithm implemented in hardware, we can get an extremely high loop rate. Once the code is downloaded to the chassis and the power is on, the system will respond within milliseconds. There is no need to know VHDL language to program FPGA; no need to be deeply familiar with real-time operating system. The above structure also uses pipelining

method to calculate 256-point FFT continuously. The use of DMA-FIFO ensures reduction in latency in display on the host and no data is lost with increased throughput.

This thesis has reported two novel and efficient architectures of split-radix FFT using NEDA. The simulation outputs of proposed architectures have not shown much deviation from numerical values. The proposed architectures are multiplier-less as well as memory-less ones. Both proposed architectures are designed for complex inputs with a data width of 16 bits, maintained constant all along. Proposed architecture – I is implemented as a fully dedicated architecture that takes all inputs in parallel and it has less delay of 4 clock cycles. But, proposed architecture – I has huge number of input-output pins; this drawback has been overcome in the later proposed architecture. Proposed architecture – II is implemented using folding which is folded so as to take 4 inputs at a time. Both the proposed architectures are implemented sequentially which results in a form of pipelining. The data flow of proposed architecture – II is clearly mentioned in table 7. Proposed architecture – II gives a maximum frequency of 527.329 MHz on Xilinx Virtex-5 FPGA and 210.97 MHz on Altera Cyclone II EP2C70 FPGA, thus showing its applicability in communication systems. There is a huge decrement in power of proposed architecture – II when compared.

The architectures for 64-point complex FFT cores using radix-4 and radix-8 algorithms have been proposed. The designs have used NEDA technique for twiddle multiplications. Both proposed architectures are designed for complex inputs with a data width of 16 bits, maintained constant all along. Both the proposed architectures are implemented sequentially which results in a form of pipelining. As compared with each other, the design using radix-4 consumes less power. But the design using radix-8 consumes less area, less number of registers and can be operated at a maximum frequency of 447.838 MHz on Xilinx Virtex-5 FPGA (327.284 MHz for design using radix-4). The slack and throughput obtained for the design using radix-8 is more than that of the design using radix-4.

The proposed architectures for 64-point complex FFT cores using radix-4 and radix-8 have also been compared to other existing designs and commercially available IP cores in terms of power and area consumption and number of clock cycles used to get the output. As shown in the comparison table the proposed design using radix-8 takes only 17 clock cycles to give the final output and it the smallest number as compared to others. The proposed design using radix-8 also consumes less area as compared to others. In the other hand, the proposed design using radix-4 consumes less power as compared to other existing designs.

The physical design of all the proposed architectures has been made in such a way that the timing constraints are met after both placement as well as routing.

6.2 Future Research

The above proposed architectures can be used in various signal processing, image processing and communication engineering applications. The complete FPGA and ASIC flow of the proposed 512-point FFT using radix-8 algorithm is to be carried out. The multiplier-less architecture can also be used in different mathematical transforms like Discrete Cosine Transform (DCT) and Wavelet Transform.

The proposed designs can still be optimized to give better results by taking care of precise shifting in the NEDA blocks. Folding technique can also be used more efficiently in the proposed architectures. Different higher radix FFT algorithms can be combined to get better results.

Last but not the least, the multiplier-less concept along with the use of folding and pipelining can be extended to various types of computation extensive applications and might result in area, power consumption, and delay reduction.

Bibliography

[1] John G. Proakis, Dimitris G. Manolakis, "Digital Signal Processing Principles, Algorithms, and Applications", Prentice-Hall, 1998.

[2] A. V. Oppenheim, R.W. Schafer, and J.R.Buck, Discrete-Time Singal Processing, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1998.

[3] P. Duhamel and M. Vetterli, "Fast Fourier Transforms: A Tutorial Review and A State of The Art," *IEEE Signal Processing Society*, vol. 4, no. 19, pp. 259 – 299, 1990,.

[4] James W. Cooley and John W. Tukey, "An Algorithm for Machine Calculation of Complex Fourier Series," *Mathematic of Computation*, vol. 19, pp. 297 – 301, 1965.

[5] Pere Marti-Puig, "Two Families of Radix-2 FFT Algorithms With Ordered Input and Output Data," *IEEE Signal Processing L*etters, vol. 16, no. 2, pp. 65 – 68, Feb. 2009.

[6] F. Arguello and E. Zapata, "Constant geometry split-radix algorithms," *Journal of VLSI Signal Processing*, 1995.

[7] Chao Cheng and Keshab K. Parhi, "High-Throughput VLSI Architecture for FFT Computation" *IEEE Transactions on Circuits and Systems - II: Express Briefs*, vol. 54, no. 10, pp. 863 – 867, oct. 2007.

[8] Manohar Ayinala, and Keshab K. Parhi, "FFT Architectures for Real-Valued Signals Based on Radix-2^3 and Radix-2^4 Algorithms" *IEEE Transactions on Circuits and Systems - I*: *Regular Papers*, vol. PP, no. 99, pp. 1 – 9, Feb. 2013.

[9] Steven G. Johnson and Matteo Frigo, "A Modified Split-Radix FFT with Fewer Arithmetic Operations," *IEEE Trans. Signal Processing*, vol. 55, no. 1, pp. 111 – 119, Jan. 2007.

[10] M. Shin and H. Lee, "A high-speed four-parallel radix-2^4 FFT processor for UWB applications," *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, pp. 960–963, 2008.

[11] Shen-Jui Huang, and Sau-Gee Chen, "A High-Throughput Radix-16 FFT Processor With Parallel and Normal Input/Output Ordering for IEEE 802.15.3c Systems," *IEEE*

Transactions on Circuits and Systems—i: Regular Papers, vol. 59, no. 8, pp. 1752 – 1765, Aug. 2012.

[12] Mario Garrido, J. Grajal, M. A. Sánchez, and Oscar Gustafsson, "Pipelined Radix-2^k Feedforward FFT Architectures," *IEEE Trans. VLSI Syst.*, vol. 21, no. 1, pp. 23 – 32, Jan. 2013.

[13] Y. Chen, Y. Tsao, Y. Wei, C. Lin, and C. Lee, "An indexed-scaling pipelined FFT processor for OFDM-based WPAN applications," *IEEE Trans. Circuits Syst. II: Exp. Briefs*, vol. 55, no. 2, pp. 146–150, Feb. 2008.

[14] M. Rawski, M. Vojtynski, T. Wojciechowski, and P. Majkowski, "Distributed Arithmetic Based Implementation of Fourier Transform Targeted at FPGA Architectures," *Proc. Intl. Conf. Mixed Design*, pp. 152 – 156, Jun. 2007.

[15] S. Chandrasekaran, and A. Amira, "Novel Sparse OBC based Distributed Arithmetic Architecture for Matrix Transforms," *Proc. IEEE Intl. Sym. Circuits and Syst*., pp. 3207 – 3210, May 2007.

[16] Richard M. Jiang, "An Area-Efficient FFT Architecture for OFDM Digital Video Broadcasting," *IEEE Trans. Consumer Elect*., vol. 53, no. 4, pp. 1322 – 1326, Nov. 2007.

[17] Stanley A. White, "Applications of Distributed Arithmetic to Digital Signal Processing: A Tutorial Review," *IEEE ASSP Magazine*, vol. 6, no. 3, pp. 4 – 19, Jul. 1989.

[18] Wendi Pan, Ahmed Shams, and Magdy A. Bayoumi, "NEDA: A New Distributed Arithmetic Architecture and its Application to One Dimensional Discrete Cosine Transform," *Proc. IEEE Workshop on Signal Processing Syst*., pp. 159 – 168, Oct. 1999.

[19] Roberto Sarmiento, F´elix Tobajas, Valent´ın de Armas, Roberto Esper-Cha´ın, Jos´e F. L´opez, Juan A. Montiel-Nelson, and Antonio N´u˜nez, "A CORDIC Processor for FFT Computation and Its Implementation Using Gallium Arsenide Technology," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 6, no. 1, pp. 18 – 30, Mar. 1998

[20] S.S. Abdullah, Haewoon Nam, M. McDermot, J.A. Abraham, "A High Throughput FFT Processor With No Multiplier," *IEEE International Conference on Computer Design (ICCD 2009)*, pp. 485 – 490, Oct. 2009.

[21] Ren-Xi Gong, Jiong-Quan Wei, Dan Sun, Ling-Ling Xie, Peng-Fei Shu and Xiao-Bi Meng, "FPGA Implementation of a CORDIC-based Radix-4 FFT Processor for Real-Time Harmonic Analyzer," *Intl. Conf. on Natural Computation*, pp. 1832 – 1835, Jul. 2011.

[22] Keshab K. Parhi, "VLSI Digital Signal Processing Systems: Design and Implementation", Wiley, 1999.

[23] Manohar Ayinala, Michael Brown, and Keshab K. Parhi, "Pipelined Parallel FFT Architectures via Folding Transformation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 6,pp. 1068 – 1081, Jun. 2012.

[24] J. Agust´ın Rodr´ıguez, Pedro M. Juli´an, Andreas G. Andreou, "Frame and arithmetic pipelining for a radix-4 FFT streamed core," *Argentine School of Micro-Nanoelectronics Technology and Applications (EAMTA)*, pp. 112 - 116, Oct. 2010.

[25] N. Mahdavi, R. Teymourzadeh, *IEEE Student Member*, Masuri Bin Othman F, "VLSI Implementation of High Speed and High Resolution FFT Algorithm Based on Radix 2 for DSP Application," *5th Student Conference on Research and Development (SCOReD 2007)*, pp. 1 – 4, Dec. 2007.

[26] Marek Horinek, Petr Bilik, "Power Analyzer for Converter Testing Based on Crio Hardware Platform," *Applied Electronics International Conference*, pp. 1 - 4, 2010.

[27] Tao Lin, Yongxing Xie, Jing Tang, "Design of CompactRIO-based Acquisition System," *Conference on Environmental Science and Information Application Technology*, Vol. 1, pp. 678 - 681, 2010.

[28] Carroll Dase, Jeannie Sullivan falcon, Brain Maccleery, "Motorcycle Control Prototyping Using an FPGA-Based Embedded Control System," *IEEE Control Systems Magazine*, pp.17-21, Oct. 2006.

[29] Maciej Rosol, Adam Pilat, Andrzej Turnau, "Real-time controller design based on NI Compact-RIO," *Proceedings of the International Multiconference on Computer Science and Information Technology*, pp. 825–830, Oct. 2010.

[30] P. Duhamel and H. Hollmann, "Split-radix FFT algorithm," *Electron. Lett.*, vol. 20, no. 1, pp. 14 – 16, Jan. 1984.

[31] Mingxi Sun, Liyu Tian and Dongmin Dai, "Radix-8 FFT Processor Design Based on FPGA," *5th International Congress on Image and Signal Processing (CISP)*, pp. 1453 – 1457, Oct. 2012.

[32] Saad Bouguezel, M. Omair Ahmad, and M.N.S. Swamy, "Improved Radix-4 and Radix-8 FFT Algorithms," *Proceedings of the 2004 International Symposium on Circuits and Systems, 2004. (ISCAS '04)*, vol. 3, pp. III - 561-4, May 2004.

[33] Details about CompactRIO, National Instruments, [Online].
Available: http://www.ni.com/compactrio/

[34] Details about CompactRIO 9104, National Instruments, [Online].
Available: http://sine.ni.com/nips/cds/view/p/lang/en/nid/203500

[35] Details about CompactRIO 9201, National Instruments, [Online].
Available: http://sine.ni.com/nips/cds/view/p/lang/en/nid/208805

[36] Details about CompactRIO 9263, National Instruments, [Online].
Available: http://sine.ni.com/nips/cds/view/p/lang/en/nid/208806

[37] P. Duhamel, "Implementation of split-radix FFT algorithms for complex, real, and real-symmetric data," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-34, pp. 285 – 295, Apr. 1986.

[38] Jes´us Garc´ıa, Juan A. Michell, and Angel M. Bur´on, "VLSI Configurable Delay Commutator for a Pipeline Split Radix FFT Architecture," *IEEE Trans. On Signal Processing*, vol. 47, no. 11, pp. 3098 – 3107, Nov. 1999.

[39] Cynthia Watanabe, Carlos Silva, and Joel Muñoz, "Implementation of Split-Radix Fast Fourier Transform on FPGA," *Proc. Programmable Logic Conference*, vol. 6, pp. 167 – 170, Mar. 2010.

[40] Xilinx Product Specification, High-Performance 64-Point Complex FFT/IFFT V1.0.5 [Online]. Available: http://www.xilinx.com/ipcenter

[41] Altera Megafunction Technical Brief ISS High-Performance 64-Point FFT/IFFT [Online]. Available: http://www.spinnaker.co.jp/jp/datasheet/tb_fft_1_V002.pdf

[42] Product Design Data Sheet, FFT-1024 Complex 1024-Points FFT/IFFT Processor. Icomm Tech. Inc. [Online].Available: www.icommtech.com/Products/FFT- 64.PDF

[43] M3 Architecture Specification Rev. 1.3: Functional Specification, Technische Universität, Dresden, Germany, 1999.

[44] Koushik Maharatna, Eckhard Grass, and Ulrich Jagdhold, "A 64-Point Fourier Transform Chip for High-Speed Wireless LAN Application Using OFDM," *IEEE J. Of Solid-State Circuits*, vol. 39, no. 3, pp. 484 – 493, Mar. 2004.

[45] X. Wang, and Y. Liu, "Special-purpose computer for 64-point FFT based on FPGA", *International Conference on Wireless Communication & Signal Processing*, pp. 1-3, November 2009.

[46] H. Son, S. Lee, and K. Min, "FPGA implementation of UWB radar signal processing for automative applications", *2010 European Wireless Technology Conference*, pp. 49-52, September 2010.

[47] J. V. McCanny, D. Trainor, Y. Hu, and T. J. Ding. Rapid design of complex DSP cores. [Online]. Available: http://www.esscirc.org/papers-97/102.pdf

[48] T. Chen and L. Zhu, "An expandable column FFT architecture using circuit switching network," *J. VLSI Signal Process.*, vol. 6, no. 3, pp. 243–257, Dec. 1993.

[49] T. Chen, G. Sunanda, and J. Jin, "COBRA: A 100-MOPS single-chip programmable and expandable FFT," *IEEE Trans. VLSI*, vol. 7, pp. 174–182, June 1999.

[50] C.W. Hui, T. J. Ding, and J. V. McCanny, "A 64-point Fourier transform chip for video motion compensation using phase correlation," *IEEE J. Solid-State of Circuits*, vol. 31, pp. 1751–1761, Nov. 1996.

[51] Wei Han, T. Arslan, A.T. Erdogan, M. Hasan, "A novel low power pipelined FFT based on subexpression sharing for wireless LAN applications," *Proc. IEEE Workshop on Signal Processing Systems*, pp. 83-88, 2004.

[52] M. Hasan, T.Arslan, and J.S. Thompson, "A novel coefficient ordering based low power pipelined radix-4 FFT processor for wireless LAN applications," *IEEE Transactions on Consumer Electronics*, vol. 49, no. 1, pp. 128-134, Feb. 2003.

[53] Song-Nien Tang, Chi-Hsiang Liao, Tsin-Yuan Chang, "An Area- and Energy-Efficient Multimode FFT Multimode FFT Processor for WPAN/WLAN/WMAN Systems," *IEEE J. Of Solid-State Circuits*, vol. 47, no. 6, pp. 1419 – 1435, Jun. 2012.

[54] Indranil Hatai, Rakesh Biswas2, Swapna Banerjee3, "ASIC Implementation of a 512-point FFT/IFFT Processor for 2D CT Image Reconstruction Algorithm," *IEEE Students' Technology Symposium (TechSym)*, pp. 220 – 225, Jan. 2011.

[55] Tanvir Ahmed, Mario Garrido, and Oscar Gustafsson, "A 512-point 8-parallel pipelined feedforward FFT for WPAN," *Conference Record of the Forty Fifth Asilomar Conference on Signals, Systems and Computers (ASILOMAR)*, pp. 981 – 984, Nov. 2011.

Dissemination of Work

[1] Ansuman DiptiSankar Das and K. K. Mahapatra, "Real-Time Implementation of Fast Fourier Transform (FFT) and Finding the Power Spectrum Using LabVIEW and CompactRIO", *Conference on Communication Systems and Network Technologies (CSNT)*, 2013 (accepted by IEEE).

[2] Ansuman DiptiSankar Das, Abhishek Mankar, N Prasad, K. K. Mahapatra, and Ayas Kanta Swain, "Efficient VLSI Architectures of Split-Radix FFT using New Distributed Arithmetic", *International Journal of Soft Computing and Engineering (IJSCE)*, vol. no. 3, issue 1, pp. 264 - 271, Mar. 2013.

[3] Abhishek Mankar, Ansuman DiptiSankar Das and N Prasad, "FPGA Implementation of 16-Point Radix-4 Complex FFT Core Using NEDA", *2nd Students' Conference on Engineering and Systems (SCES)*, 2013 (accepted by IEEE).

[4] Abhishek Mankar, N Prasad and Ansuman DiptiSankar Das, "FPGA Implementation of Retimed Low Power and High Throughput DCT Core Using NEDA", *2nd Students' Conference on Engineering and Systems (SCES)*, 2013 (accepted by IEEE).

[5] Abhishek Mankar, N Prasad, Ansuman DiptiSankar Das, Sukadev Meher, "Multiplier-less VLSI Architectures for Radix-2^2 Folded Pipelined Complex FFT Core," *International Journal of Circuit Theory and Applications* (Communicated to IJCTA), Mar. 2013.

[6] Ansuman DiptiSankar Das, Abhishek Mankar, N. Prasad, K. K. Mahapatra, "Multiplier-less VLSI Architecture for Radix-8 Folded Pipelined Complex FFT Core," *The CoreEL-Digilent Design Contest 2013* (Selected for further proceedings among top 20).